Carderock Division
Naval Surface Warfare Center
Bethesda, Maryland 20084-5000

METAFILE FOR INTERACTIVE DOCUMENTS, VERSION 2

Application Guide and Draft Performance Specification for the Encoding of Interactive Documents - MID-2 (3/96)

March 1996

Eric L. Jorgensen, Research and Technology Representative
Navy Technical Manual Working Group

Prepared by the MID-2 Design and Development Team:
Darlene Janiszewski, NAWC-AD, Project Manager
L. John Junod, NSWC-CD
Michael Anderson, Antech Systems, Editor and Team Chairman
David Cooper, Antech Systems

Note: This final draft of the MID-2 (3/96) Specification and Application Guide has been prepared by the U. S. Navy for purposes of review and comment by the general Navy, DoD, National, and International technical community interested in standards for Interactive Electronic Documents which require a mechanism (i.e., script) for controlling the presentation of text, graphics, and other multimedia information developed for electronic display. It is written as an application of ISO 8879 SGML and utilizes portions of ISO 10744 HYTIME. While it was initiated by the Navy for purposes of developing a run-time standard for DoD Interactive Electronic Technical Manuals (IETMs), the MID-2 (3/96) draft specification has been intentionally developed to be suitable for application to generic scripted interactive documents of any nature and for any application. The Navy point of contact regarding potential specification coordination issues is Eric Jorgensen, NSWC-CD Code 182, email: jorgense@oasys.dt.navy.mil.

The Metafile for Interactive Documents, Version 2
Application Guide and Draft Performance Specification
for the Encoding of Interactive Documents
MID-2 (3/96)

March 1996

Prepared by the MID-2 Design and Development Team

Project Manager:
Darlene Janiszewski, NAWC-AD St. Inigoes

Team Chairman & Document Editor:
Michael Anderson, Antech Systems

Team Members:

Len Bullard, Loral
Terri Castelli, CSC
David Cooper, Antech Systems
Michael Croswell, CSC
Mark Drissel, CSC
Rob Groat, Booz Allen
Eric Jorgensen, NSWC Carderock
L. John Junod, NSWC Carderock
Neill Kipp, TechnoTeacher
Steve Newcomb, TechnoTeacher
Perry Rapp, CSC
Rob Sommerville, CSC

Contents

1. How to use this Application Guide

The primary purpose of this document is to communicate, to authors and developers of MID documents and applications, the intentions of the design team with respect to implementation of the MID DTD. Section 6 contains the DTD, with annotations, arranged by functional groupings of elements as described in Section 5. The MID design team has debated many issues in creating this DTD, and the annotations are presented to pass along as much of that consideration as possible. There are some issues remaining, and the MID project staff solicits your input to making the design more useful and complete.

Note that the use of terms such as “specification” or “standard” are not intended to claim the sanctioning of MID in an official sense. The terminology for describing MID is intended to convey its purpose, namely as a common structure into which technical information can be translated. For the record, the MID is the result of a research effort, not the work of a standards body. It is our hope that the technology and techniques employed by MID will prove the concepts to be valid, and provide guidance to those who would solve the problem of transporting information and logic between source providers and presentation software. Adoption of MID as an official specification or standard is a consideration for the future.

This document also contains a general introduction to the MID (Sections 3 & 4), and references for more information and background (Section 2). For the seasoned MID user, Section 7 provides an alphabetical reference to the elements, and Section 8 chronicles the changes from the original MID definition (1994) to the current, evolutionary version (1995). Appendix A contains a processable MID DTD. Appendix B contains an example using elements derived from the **relationship** architectural form defined by the MID. Appendix C contains excerpts from the MID Draft Specification, published in 1994, as an introduction to the original MID concepts and goals.

2. References and Sources for Additional Information

2.1 References

MID Draft Specification. Carderock Division, Naval Surface Warfare Center, Nov. 1994

2.2 Other sources

2.2.1 NAWC-AD, Patuxent River MD, Applied Technology Branch, Code 4.5.8.6

The R&D project that has resulted in the MID concept, design, and prototype implementation was initiated by the Naval Air Warfare Center - Aircraft Division, St. Inigoes, MD.

The Navy Project Manager is Ms. Darlene Janiszewski. For information concerning the current status of the project, or for collaboration among Navy projects interested in IETM or MID development, contact Ms. Janiszewski by email at <djaniszews@ietm.nawcsti.navy.mil>.

2.2.2 NSWC Carderock

The MID project has been coordinated through the Navy representative on the Tri-Services IETM Working Group, the Naval Surface Warfare Center - Carderock Division. Representatives at Carderock have been integrally involved in setting priorities, identifying technical issues, and coordinating among IETM development programs during the MID concept and design phases.

For information concerning the relationship of MID to other Navy and DoD IETM development projects, and to find out more about the IETM Tri-Service Specifications (MIL-D-87269 and MIL-M-87268), contact Mr. John Junod at <junod@oasys.dt.navy.mil>.

2.2.3 MID Design & Development Team Members

The MID Design Team was formed in 1994 to consider possible solutions for problems with IETM interoperability. The Team identified the problem as being primarily related to transport of IETM data between various presentation systems. The MID design grew out of a series of meetings where the team considered a wide range of possible technical solutions.

During 1995, the focus changed from identifying and formulating the basic technical approach for IETM data transport, to proving and improving the technical design through implementation. A software development effort was launched, as well as a series of analysis tasks aimed at integrating the MID approach with related, existing and emerging, standards and technologies. The development and analysis efforts both have the purpose of evolving the MID design.

The following MID Design Team members were involved as indicated, and may be contacted via email (where listed) for further information.

Member	Organization	Involvement	Email
Michael Anderson	Antech Systems	1995 chairman 1994 design	antech@norfolk.infi.net
Vince Botticelli	Lockheed Ft Worth	1994 design	
Len Bullard	Loral	1995 analysis 1994 design chairman	cbullard@HiWAAY.net
Bryan Caporlette	Passage Systems	1994 design	
Terri Castelli	CSC	1995 development	
David Cooper	Antech Systems	1995 development lead 1994 design	dwcooper@nando.net
Michael Croswell	CSC	1995 development	
Mark Drissel	CSC	1995 development	
Rob Groat	Booz Allen	1995 development	
Darlene Janiszewski	NAWC-AD St. Inigoes	1995 program management 1994 program management	djaniszews@ ietm.nawcsti.navy.mil
Eric Jorgensen	NSWC Carderock	1995 Technical review 1994 Technical review	
L. John Junod	NSWC Carderock	1995 analysis 1994 design	junod@oasys.dt.navy.mil
Neill Kipp	TechnoTeacher	1995 development 1994 design	neill@techno.com
Steve Newcomb	TechnoTeacher	1995 development 1994 design	srn@techno.com
Mark Petronic	Hughes Aircraft	1994 design	
Perry Rapp	CSC	1995 development	
Rob Sommerville	CSC	1995 development	
Madeleine Sparks	Loral	1994 admin.	

3. MID Definition

The Metafile for Interactive Documents (MID) is a common interchange structure, based on the international standards for SGML and HyTime, that takes neutral data from varying authoring systems and structures it for display on dissimilar presentation systems. [*MID Draft specification, Nov. 94*]. It is envisioned that a MID instance will be a hub document, containing references to various, external source data components. The MID instance will be created by an interactive, automated process (i.e., a "MIDWriter"), and will be interpreted for viewing by off-the-shelf software incorporating a "MIDReader."

Development of a MIDReader was the primary focus of the 1995 MID project, and its development has served to both highlight issues in the structure of the MID, and identify implementation issues. Resolution of these issues has resulted in an evolutionary improvement to the MID specification.

The MID definition is directed to solving a well-known and pervasive problem in the IETM development community: moving IETM data between presentation products while preserving the (critically important) logic coded in the data. A major goal of the effort has been to support the Tri-Service IETM Specifications for data storage (MIL-D-87269), and enable compliance with presentation standards such as MIL-M-87268. The MID makes extensive use of SGML and HyTime to accomplish this goal in an open and extendible architecture. The following diagram illustrates the intended use of a MID instance in the context of an IETM delivery.

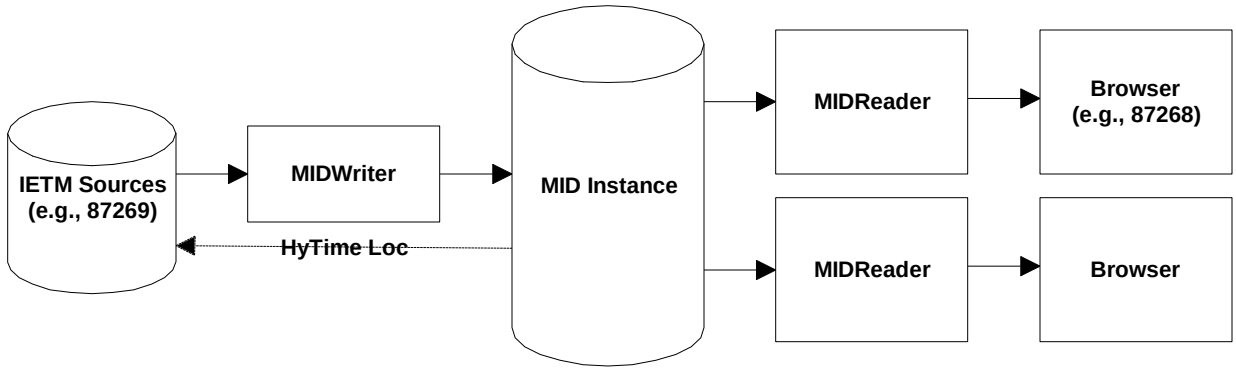


Figure 1: MID Architecture Overview

MID does not contain format or style information. MID browser software will have to determine positioning and appearance of MID elements, either explicitly or through some intermediate structure. Document Style Semantics and Specification Languare (DSSSL) appears to be a likely candidate to apply style and presentation semantics to the MID definition. However, until an authoritative list of the style characteristics appears, each MIDReader/ browser application will be responsible for defining how to handle the presentation appearance without coding it into the MID instance. A good option would be to design SGML structures to specify application of style. This should allow maximum flexibility in adopting standard methods when they are available, e.g., through tree transforms to DSSSL.

4. Why you need MID

4.1 Behavior of applications

In the context of Interactive Electronic Technical Manuals (IETMs), 'Interactive' means that the application reacts to input from users on a real time basis. This reaction is often to tailor the content and presentation of subsequent information.

To create an IETM, authors and developers must consider a philosophy different from that used to create page-based documents; they must program behavior into the document. The encoding of logic to control document behavior is one facet of electronic delivery that can cause incompatibility between IETM (and other) information systems. The development of IETM Specifications for document delivery, such as MIL-D-87269, standardize the structures for IETM content, and introduce logical conditions for information rendering. The MID adds the missing layer - standardization of the methods for encoding document behavior, and connecting the content to presentation in an unambiguous way.

4.2 The MID approach to standardizing document behavior

The MID, as its name implies, uses a hybrid metafile approach to define templates and methods for encoding logic intermixed with information content. In the classic sense, a metafile in SGML provides a template that guides an

author in creation of a DTD. The MID uses "meta" definitions for linking of information, but also defines structures to be used as translation targets for information to be rendered, and structures for specifying logical flow.

The way that the MID specifies logical flow is through SGML constructs that are borrowed from common programming languages. Just as compilers and interpreters require a standard syntax to create executable software from a programmer's code, a MIDReader requires a standard syntax to produce an interactive presentation from an IETM author's document.

In a paper technical manual, an author makes assumptions about the projected level of expertise of the technician, and then provides tables, diagrams, and paragraphs of text that address, for example, each of the steps in a repair procedure. Steps that are only done under certain conditions are mixed with steps that are *always* applicable, because there is no mechanism for turning off the display of steps that are not applicable. An IETM, on the other hand, can use multimedia output to show only the applicable information. The application decides when to render (e.g., display) this information, and how to organize it for maximum efficiency.

The MID structures are what enable an application to determine when, and nominally where, to render information. The decisions as to when information gets rendered are made by decoding the logic in a MID script, which in turn may reflect the logic embedded in the source file (MIL-D-87269). An IETM author, already burdened with the responsibility to encode logic in documents, now has a standardized way to do it. The decisions as to where information gets rendered are derived from semantic grouping of renderable elements. The MID scripting language allows:

- Conditional rendering
- Logical grouping of elements to be rendered
- Expressions, functions, statements
- Storage of values for later use (i.e., variables), and definition of where the variables apply (i.e., scoping)
- Passing of responsibility to external processes, and means for defining the parameters of the processing

These will be described further in Section 5, and in the Application Notes for individual elements in Section 6.

5. General description of theory

The following paragraphs outline the major types of elements found in the MID DTD. This description is intended only to give a general overview of how the MID accomplishes its stated goals.

5.1 Containers

InfoContainers define a set of information that is rendered as a package. In the simplest case, an **infoContainer** (IC) can define frames of *text* and *graphics* that will be simultaneously displayed. In a more complex case, the IC might contain a *script*, with a set of variables that affect its behavior, or the behavior of subsequent **infoContainers**. Transient **panes** of information, and user interactions such as **alerts** and dialogs, might be part of an IC. Conditional information can be *reflowed*, based on user input, without leaving the IC. While it is theoretically possible (i.e., syntactically correct) to place an entire IETM in a single **infoContainer** by using the power of *scripting*, this would constitute poorly-formed MID; authors are encouraged to use the IC to good advantage by logical arrangement.

InfoContainers may be arranged in **chains** for sequential presentation starting from a defined point, or placed in **pools** where they can be reused.

Panes define individual elements of content to be rendered. Typically, a browser would render the contents of a **pane** in a window of a graphical user interface. Many **panes** may be displayed as part of a single **infoContainer**. A **pane** encapsulates a scope that defines its own set of variables, and may use a *script* to retrieve content from a source document. A common use of *scripting* in a **pane** would be to *get* information content, based on some conditions, using a HyTime link.

Rendering of **panes** within given screen real estate is the purview of the browser. There are no position, size, or other visual properties defined as part of the MID **pane**. Such properties must be derived from a combination of the logical (i.e., semantic) *grouping* of the **panes**, and the application of an optional stylesheet from a source external to the MID.

Panes can be used to implement user interactions by defining a set of *controls* to be rendered on the **pane**. Again, the position and style of *controls* can not be specified in the MID. In fact, the type of *control* used in a particular software environment may vary significantly, depending on the look and feel of the operating system. The MID, for example, defines a *dynamicList* element that enumerates choices for a user selection. The list is specified by an attribute to be either of type *pickOne* or of type *pickMany*. A *pickOne* type *dynamicList* could be rendered with equal effectiveness as a drop down list box, a menu, or a set of mutually exclusive (radio) buttons.

Elements of type *Containers*	Element #	Description
infoContainer	11	The fundamental building block of a MID application, defining a logical package of information to be rendered.
chain	12	A set of **infoContainers** that are intended to act as a sequential set. A chain must be navigated starting with the first **infoContainer** in the sequence.
pane	13	The delimiter for a logical fragment of content; a component part of an **infoContainer**. The **pane** should be filled with a single reference, loc, query, or contiguous piece of information content.
alert	14	Special case of a **pane**, where the contents are to be rendered as a transient interaction that must be acknowledged by the user before continuing (modal).
clientArea	17	Container for **panes**, *paneGroups*, and *conditionalPanes*.
pool	15	A common resource area, designed to house reusable **containers**, *scripts*, and *controls*.
messageArea	16	A container for reporting supplemental information (e.g., instructions, state of the application) that would typically be related to the information content (rather than the operation of the browser software).

5.2 Transitions & Links

Transitions can be specified by an author to occur between *infoContainers*, or within an *infoContainer* to render new *panes* or *controls*. Types of transitions may include **goto**, **gosub**, or **spawn**. The difference between these transition types lies in the way the application handles the state of the currently-rendered *containers*. For example, a *button* on a *pane* within *infoContainer* IC-1 might contain a *script* that implements a **goto** transition to *infoContainer* IC-2. In this case, all non-global variables (e.g., set by the *infoContainer* or a *pane* in IC-1) would be cleared, and the state space would be set to represent IC-2 *variables*. In a similar case where a **gosub** was specified in place of the **goto**, the IC-1 *variables* would be maintained, and then restored at the completion of IC-2. Using a **spawn** would instruct the application to simultaneously maintain both sets of state *variables*. By these mechanisms, an author has the ability to control the appearance of an application, without undue restriction on the application's unique look and feel.

A **get** specifies that information at a source be retrieved, and rendered at the point of the **get**. This element represents one of the most important powers that MID offers. The **get** allows source documents, delivered and maintained independently, to be bound to the presentation at run-time. It also allows source documents to be in various formats, and still be accessible to a MID instance. The power of HyTime location and linking facilities is what makes this capability both practical and standard.

Relationship is a construct based on a HyTime ilink that allows authors to establish connections between various types of information in a document. **Relationship** may be used to implement hotspots in *text* and *graphics*. The **relationships** in a MID document are located in the **pool**.

Because **relationship** is an element pseudo-declaration, many elements may be created to implement **relationships** within a given MID. Each of the derived elements will define a specific type of **relationship**, with the instances of that element linking information that is related in the defined way. Rendering of the **relationships** is up to the application; often the anchors of the **relationship** links will be treated as hotspots that can be selected by users, and traversed according to the rules of HyTime ilinks.

Relationships are based on named connections between related information. This enables the author to specify, and the browser application (MIDReader) to determine, the reason for a particular link, in addition to the linked objects. Thus, the nature of the **relationship** is transported from source to end-user.

Elements of type *Trans & Link*	Element #	Description
goto	19	Implements a transition to a new context, and forgets about the old one.
gosub	19	Implements a transition to a new context, but keeps track of the old one so the application can return when the new context is terminated, and restore the previous state.
spawn	19	Maintains the state of two contexts simultaneously; application allows user to transition between them at will.
get	20	Enables the collection of information from remote sources, for rendering at runtime.
relationship	21	A pseudo-declaration (element template) for establishing types of links that are meaningful in a given document context. The elements created using the **relationship** template may be used for implementing hyperlinks in a browser application.

5.3 Controls

Controls define means for users to report events back to the MID instance. Authors specify what **menus** or **buttons** should be associated with a particular *infoContainer*, and how the application should respond to selection of such a control. Also, user interaction elements such as **fillin** and **dynamicList** can be combined with other controls to gather information through an interactive dialog.

Menus are intended as a generalized way for authors to define the high-level entry points into an information set. The term menu has certain implications to software developers with respect to rendering. However, menus differ from lists in the MID only by virtue of the fact that they contain a set of **buttons** rather than elements of content.

The term **button**, as **menu**, has a connotation in the context of graphical user interface (GUI) development. However, the MID concept of a **button** does not necessarily imply rendering as a graphical push-button. **Buttons** will be considered, during the rendering of a MID instance by an application, to denote the function normally performed by a **button**, namely to launch a process.

Elements of type *Controls*	Element #	Description
menu	23	A collection of **buttons** used for launching processes in a MID browser. **Menus** are grouped together into *menubars*, and reused in *infoContainers* as appropriate.

button	25	A structure representing the generalized function of launching a process.
fillin	26	A field that allows free-form user input of text.
dynamicList	27	List of content items that is built at runtime using an *expression*.

5.4 Data Types

The elements listed as data types are used for containing content information that is rendered, for example, in a *pane*. Depending on the implementation of MID, the data may be contained directly in the element, or retrieved by a *script*.

Elements of type *Data Type*	Element #	Description
text, paragraph	30, 33	Contains character data or other text items.
specialText	31	Indicates some text that is qualified by a semantic.
title	32	Defines the text that is to be rendered as a title for the element that contains it.
graphic, audio, video, animation, icon	37	These elements access external notations that define how to access various file formats.
tableType, fcsTable	39	fcsTable is a generalized method of storing data that is typically rendered in a table format.
orderedList, unorderedList, item	34	These are content intended to rendered as lists. These are not affect by the support attribute of the mid element..

5.5 Semantic Grouping

Semantic grouping is used, in lieu of coordinate systems, to indicate to applications the layout priorities for a set of rendered elements. A grouping might be used by the application to determine how to spatially allocate panes within an infoContainer, or where to put delimiters such as a 'group box' for controls within a user interaction pane. These are the only cues that the application can get directly from the MID instance to determine placement of windows and controls on the screen. Authors who do not use these structures risk placement of elements based solely on their order, or worse, random placement.

Elements of type *Sem Group*	Element #	Description
paneGroup	46	A group of panes within an infoContainer.
widgetGroup	47	A group of buttons, dynamicLists, and fillins to be used for user interaction.
buttonGroup	48	A group of related buttons that are intended for either single (pickOne) or multiple (pickMany) selection.
menubar	49	A collection of menus and buttons that is reused among infoContainers.

5.6 Conditionals

Conditionals are used in conjunction with *scripts* to indicate *panes* or *controls* that are rendered only under certain conditions. The conditions are evaluated when a **reflow** statement is encountered in a *script*.

Conditionals are particularly suited to rendering complex sets of information that contain dependencies. For example, an author may want to populate an equipment list based on the selection of equipment type. Rather than building separate *panes* with the list for each equipment type (and the requisite transitions to the proper *pane* for rendering), an author could include a *dynamicList* within a **conditionalWidget**, where the list builds its contents based on an expression that gets re-evaluated at each **reflow**.

Elements of type *Conditionals*	Element #	Description
conditionalPane	51	Wraps a paneGroup that is rendered in entirety when a reflow is encountered in a script.
conditionalWidget	52	Wraps a widgetGroup that is rendered in entirety when a reflow is encountered in a script.
reflow	53	This element allows authors to render the conditional elements within a specific target and its subtree.

5.7 Scripting

Scripting is the means for MID to incorporate application behavior in a document. Authors have control over the flow of logic by **functions**, **variables**, **statements**, **strings**, and other operations. **Scripts** must be interpreted by MID 'engine' software, which is developed as part of a browser application, to read or import a document.

Scripts are most often used to determine how, when, and where to get information for rendering. Thus, **scripts** are usually closely associated with *transition* and *link* elements such as *goto*, *gosub*, and *spawn*.

Elements of type *Scripting*	Element #	Description
script	55	Declares an element which encapsulates the logic defining behavior of a document.
name	56	Identifies a function, variable, xenodecl, or scriptLabel for purposes of maintaining system state during execution of a script.
statements	57	A convenient wrapper for logical processes available for use in a script.
expr	65	A statement type that is evaluated by a script interpreter, and returns a copy of the result.
assign	67	Places the results of an evaluated expression in a variable.
vardecl / variable	59/ 66	Declares / stores a value.
funcdecl / function	60/ 68	Declares a function / Sends arguments to a funcdecl or xenodecl for evaluation.
argdecl / argument	61/ 69	Declares an argument / Passes the results of an expression to a function for use by the function.
constant	71	Stores a value of a given type.
stringOperations	63	A parameter entity enumerating the types of string operations in a MID document.
listOperations	64	A parameter entity enumerating the types of list operations in a MID document.
if, else, loop, continue, break, switch, case, default, jump, scriptLabel, gettype	72, 73, 74, 75, 76, 76, 77	Structures that specify flow control in a script. These generally specify execution of a statement based on evaluation of an expression.
add, multiply, subtract, divide, modulus, eq, lt, gt, le, ge, and, or, ne, not	70	Represent operations to be performed on one or more expressions, as indicated.

5.8 External Processes

External processes are accommodated through SGML notations and a structure called a **xeno**. The **xeno** element type definition declares an element which declares a function written in non-SGML encoding and processed external to the MID. That is, it is used in the same way that an assembler or other non-native application language can be declared from within another language, e.g., C or ADA. These processes could also be used, for example, to tell a browser how to use a graphics server to display a particular format of graphics. The **xeno** requires that a notation location be specified to enable the MID application to determine which external process must be called to handle the declared function when called and what arguments must be passed to it.

Elements of type *Ext Process*	Element #	Description
xenodecl	95	Binds a name and argument declarations to a call to an external notation.
xeno	96	Declares an element that calls a function that is not coded in SGML.

5.9 HyTime Location and Linking Constructs

MID uses many HyTime constructs to provide location and naming conventions, and to link related information together for retrieval by dissimilar presentation systems.

Elements of type *HyTime*	Element #	Description
nameloc, nmlist	101, 102	Named location addresses.
treeloc, relloc, dataloc	104, 105, 106	Coordinate location addresses.
proploc, notloc, bibloc	109, 115, 116	Semantic location addresses.

5.10 HyTime and SGML Management

5.10.1 HyTime Module Declarations

HyTime module declarations that declare which features of the metalanguage standardized by ISO -10744 must be supported for a MID document entity must be included in the prolog of an SGML document. They follow the close of the SGML Declaration.

A MID requires the use of the following HyTime modules:

<?HyTime VERSION "ISO/IEC 10744:1992" HYQCNT=32>

<?HyTime MODULE base exidrefs>

<?HyTime MODULE measure>

<?HyTime MODULE locs anydtd coordloc HyQ multloc query relloc>

<?HyTime MODULE links manyanch>

5.10.2 SGML and HyTime Notation Declarations

The following notation declarations are required for the MID.

- SGML -- enables access to external documents encoded in SGML

  <!NOTATION SGML PUBLIC "+//ISO 8879:1986//NOTATION

  Standard Generalized Markup Language//EN">

- HyTime -- enables access to external documents encoded in the Hypermedia Time-based Structuring Language

  <!NOTATION HyTime PUBLIC "+//ISO/IEC 10744:1992//NOTATION

  Hypermedia/Time-based Structuring Language//EN">

- HyQ -- enables use of documents that include HyQ queries written in the notation prescribed by ISO 10744

  <!NOTATION HyQ PUBLIC

  "+//ISO/IEC 10744:1992//NOTATION HyTime Query Notation//EN" >

NOTE: As other non-SGML data types are required for MIDs, this list must be extended. At this time, the notations for graphics, audio, video and animation are not prescribed. Also, any notations required for accessing external processes of other types (e.g., diagnostic modules) are not defined. The HyQ HyTime query language is being merged with Standard Document Query Language, SDQL, which will also be used in the Document Style, Sematics, and Specification Language (DSSSL) Standard (ISO/IEC 10179). The NOTATION for HyQ is subject to revision based on the adoption of SDQL.

5.10.3 MID Parameter Entities

The following entities are defined for inclusion in MID element type declarations (link and loc) and attribute list declarations (yesorno only):

<!ENTITY % yesorno "NUMBER" >

<!ENTITY % loc "nameloc | treeloc | dataloc | notloc | proploc | relloc | bibloc">

5.10.4 MID Document Type Declaration

This is the formal document type declaration for a MID:

<!DOCTYPE mid PUBLIC "-//USA-DOD//DTD MID Document Type Definition 19951201//EN">

5.10.5 MID Short Reference Maps

It is recommended that the requirements of FIPS 152 which preclude the use of the shortref feature of SGML be waived for the MID. Furthermore, it is recommended that a shortref map be constructed that simplifies the entering of complex MID structures such as those used in the <script> element type statements.

6. DTD with annotations for developers and element numbers

Ref	MID DTD	Application Notes
1	<!-- MID: Metafile for Interactive Documents Document Type Definition This document type definition shall be identified by the following declaration: PUBLIC "-//MID//DTD MID Document Type Definition//EN" -->	
2	<!-- NOTATIONS -->	
3	<!NOTATION SGML PUBLIC "+//ISO 8879:1986//NOTATION Standard Generalized Markup Language//EN" > <!NOTATION HyTime PUBLIC "+//ISO/IEC 10744:1992//NOTATION Hypermedia/Time-based Structuring Language//EN" > <!NOTATION HyQ PUBLIC "+//ISO/IEC 10744:1992//NOTATION HyTime Query Notation//EN" > <!NOTATION virspace PUBLIC "+//ISO/IEC 10744:1992//NOTATION Virtual Measurement Unit//EN" >	• HyQ is to be merged with the Standard Document Query Language (SDQL), also used in the Document Style, Sematics, and Specification Language (DSSSL) Standard (ISO/IEC 10179).
4	<!-- ENTITIES -->	
5	<!-- locs These are the HyTime Location addresses used by the MID. --> <!ENTITY % locs "nameloc | treeloc | dataloc | notloc | proploc | relloc | bibloc" >	• Modification of entities is discouraged, because it will invalidate the standard. The application will not be expected to reflect modified entities in its implementation of gettype.
6	<!-- primitives, functionType, variableType, constantType The MID script primitives are listed in the entities below. The application may choose to override these declarations to extend or constrain the MID definition. An atom is string or any one of the primitives. --> <!ENTITY % primitives "boolean | int32 | uint32 | int64 | uint64 | float32 | float64 | sgmlchar" >	• Authors are recommended against using int64 and uint64 until such time that it becomes standard in ANSI C. Declare by support policy if required. • Definition of the casting rules: from uint to int, 32 to 64, int to float, char to string, int to boolean, float to boolean, boolean arithmetic: + is 'or', * is 'and'. Anything else is considered poorly-formed MID, and may result in unpredictable results.

		• Logical operators: not, and, & or only apply to booleans. • Booleans should not be used as numerics in arithmetic operations. The results of using a boolean in an arithmetic operation (add, multiply, subtract, divide, modulus) is implementation dependent, and applications are encouraged to warn of a poorly-formed MID document.
7	<!ENTITY % functionTypes "%primitives | string | atom | list | any | void" > <!ENTITY % variableTypes "%primitives | string | atom | list | any" > <!ENTITY % constantTypes "%primitives | string" >	• Assignment of a string to a character gets the first character. Assignment of character to string makes a string of unit length. • Authors should not perform arithmetic operations (add, multiply, subtract, divide, modulus) on strings or characters. This will eliminate problems with casting in operations where strings and chars are mixed. Applications will not support such operations consistently. • The use of the function type or varible type “list” will require the used the MID Support attribute. See element #9, page 13.
8	<!-- MID -->	
9	<!-- mid The mid element is the root element for a MID application. The vardecls, funcdecls, and xenodecls in its immediate content are global declarations to the MID application. The MID instance is processed by first processing the global declarations and then the master script. The MID returns the results of evaluating its master script. The type of the resulting data is given as the value of the functionType attribute. This specification is redundant and is made solely for convenience. It is a reportable MID error (RME) if the type of the return value of the master script does not match the return value of the MID. Date and version hold human-readable strings for specifying the date and version of this document. The docmdu attribute specifies the measurement domains of the document's finite coordinate spaces (fcs) and the least common unit for computing dimensions in each fcs. The security attribute identifies the security designation for this MID document. Security is implemented as a HyTime activity policy. The following are support options for the support statement. The names may be listed in any order.	

```
conditionalPane
  This document may contain conditional panes.

conditionalWidget
  This document may contain conditional widgets.

fcsTable
  This document may contain MID fcs tables.

list
  If list is specified, this document may use the list data structure
and the list expressions.  If list is not specified, list must be
deleted from the functionTypes and variableTypes parameter entities.

MIL-M-87268
  This document is intended to be used in connection with software
whose user interface strictly conforms to MIL-M-87268.

nonMID
  This document may contain addresses of locations in external
SGML/HyTime documents which are not MID documents.

query
  This document may contain queries which address locations in
external SGML/HyTime documents which are not MID documents.  If query
support is specified, nonMID support is implied.

relationship
  This document may contain MID relationship forms.

 spawn
  If spawn is specified, this document may contain spawn elements.

string
  If string is specified, this document may use the string data
structure and the string expressions.  If string is not specified,
string must be deleted from the functionTypes and variableTypes
parameter entities.

xeno
  If xeno is specified, this mid document may use the xenodecl and
xeno elements.  All data content notations must be declared using
notation declarations in the DTD.

-->


<!ELEMENT mid - O (( vardecl | funcdecl | xenodecl)*, script, pool?) >
<!ATTLIST mid
  HyTime NAME #FIXED HyDoc
  id ID #IMPLIED
  functionType (%functionTypes;) atom
  date CDATA #IMPLIED
  version CDATA #IMPLIED
```

	docmdu CDATA #FIXED "virspace 1 1" HyNames CDATA "activity security" security IDREFS #IMPLIED support NAMES "conditionalPane conditionalWidget fcsTable list MIL-M-87268 nonMID query relationship spawn string xeno" >	
10	<!-- CONTAINERS -->	
11	<!-- infoContainer When an infoContainer is accessed, its declarations are processed. The menubar is built from the list of menubars given in the attribute value, then the script (which may contain adjustments to the menubar in setState commands) is executed. After this, the title, alerts, and clientArea are processed in the order they appear. The functionType attribute specifies the return type of this construct.--> <!ELEMENT infoContainer - O ((vardecl | funcdecl | xenodecl)*, script, title?, alert*, clientArea, pool?) > <!ATTLIST infoContainer id ID #IMPLIED menubar IDREFS #IMPLIED functionType (%functionTypes;) atom >	• InfoContainers and other initialization scripts are processed in the order that they appear in the MID. For example, scripts within an infoContainer, title, alert, clientArea, etc. are processed as the elements are rendered. Button scripts are only rendered when they are activated by returned events. • InfoContainer pools contain panes or widgets that are used conditionally (or as pop-ups) within the infoContainer. They are not accessible from outside the infoContainer. • 'functionType' applies when the infoContainer is for the purpose of getting input from users.
12	<!-- chain Access to infoContainers within a chain is restricted to infoContainers within that chain. When a chain is accessed, its first contained infoContainer is processed. --> <!ELEMENT chain - O (infoContainer)* > <!ATTLIST chain id ID #IMPLIED >	• The intent is for chains to be used for sequential access of infoContainers, as in a procedure. Chains must be entered at the first infoContainer; however, once in a chain, the author can specify any order of traversal within the chain. • If the author intends that a chain be rendered, the link should be made to the chain rather than explicitly to the first infoContainer in the chain. • Spawns to other infoContainers inside the same chain are considered poorly-formed MID. Spawns outside the chain are OK.
13	<!-- pane A pane is a single user interface presentation, which is rendered when it is encountered. A pane encapsulates a scope. A get within a pane causes the target to be rendered on this pane. Scripts within panes are run when the pane is rendered. The return value of the script is the return value of the pane. It is a RME if the type of the pane and the containing script are not the same. A pane is modeless when contained in a client area or when called from spawn. A pane is modal when called from gosub. The security attribute identifies the security designation for this pane. Security is implemented as a HyTime activity policy.	

	-->
	
	<!ELEMENT pane - O ((vardecl | funcdecl | xenodecl)*, title?,
	(text | %tableTypes; | graphic | audio | video | animation | widgetGroup
	| get | script)) >
	<!ATTLIST pane
	  id ID #IMPLIED
	  functionType (%functionTypes;) atom
	  HyNames CDATA "activity security"
	  security IDREFS #IMPLIED
	>	

14	<!-- alert
	  An alert represents a modal popup window with the contained information. The contents of the alert are evaluated and rendered in the order they are encountered. The alert is popped down when a return alert statement is encountered in the button script. The type attribute indicates the semantic of the alert. -->
	
	<!ELEMENT alert - O (title?, icon*, text, button) >
	<!ATTLIST alert
	  id ID #IMPLIED
	  type (warning | caution | note) note
	>
		• Alert border styles are a functional of the rendering actions of the reader and might be specfied by use of a style sheet. The style of alert border specified in MIL-M-87268 are not a function of the MID.

15	<!-- pool Elements in the pool are not rendered until they are requested by identifier reference. The scope of all resolution of variables, etc., is always specified lexically, i.e., variables referenced in the pool are valid or invalid with respect to the containing scope (mid or infoContainer), not with respect to the caller's state. Among other things, the pool may contain elements of the following types: HyTime location address, HyTime hyperlink (e.g., relationship), chain, infoContainer, menubar, pane, alert. --> <!ELEMENT pool - O ANY >	• There are two types of pools: MID pools, and infoContainer pools. • InfoContainer pools can contain panes, widgetGroups, and other elements that can legally be contained in an infoContainer. Other infoContainers, chains, and scripts are not allowed in infoContainer pools; only in MID pools. Pools may not be contained within other pools. Authors must guard against nesting pools, e.g., within a script in a button that is in a pool. • Scripts should not be directly in pools, because there is no specified way to handle it in the DTD. • The pool is intended as a data container for reusable elements. Its lexical scope is defined by the scope of who is referencing it (i.e., pools have no impact on scope). Handle as macro substitution. • While an infoContainer could, within the rules of syntax and without harm to scoping rules, be placed in the pool of another infoContainer, this would be considered poorly-formed MID. • The script interpreter or compiler in a MIDReader doesn’t need to parse the pools directly - the

		contents of the pool are only parsed when referenced directly. This fact makes the content model of #ANY manageable for application developers.
16	<!-- messageArea The contents will be evaluated and concatenated in the order in which they appear, and the results will be rendered by the application as a status message. --> <!ELEMENT messageArea - O (get | expr | #PCDATA)* >	• 'get', as an element of messageArea, should point to #PCDATA, #RDATA, #CDATA, or to a notation.
17	<!-- clientArea A client area is the container for panes, paneGroups, and conditionalPanes. The panes are rendered in the order they are encountered. --> <!ELEMENT clientArea - O (pane | paneGroup | conditionalPane | alert)* >	• In an 87268 implementation, the last child of a client area must be a widgetGroup pane. This pane is the footer bar. The members of the widgetGroup may only be buttons. The label for the widgetGroup will not be rendered. • It should noted that an empty clientarea is allowed, which would provide for script only processing within the infocontainer.
18	<!-- TRANSITIONS & LINKS -->	
19	<!-- gosub, goto, spawn Expresses a HyTime hyperlink with specific MID script traversal semantics. Gosub indicates that the state of the current infoContainer be saved and the target object rendered. Gosub targets may be of the following types: infoContainer, chain, pane, conditionalPane, alert, mid. Gosub is forbidden to an infoContainer that is nested in another chain. Gosub is forbidden to a pane or conditionalPane that is nested in another infoContainer's client area or paneGroup. Goto indicates that the current infoContainer be abandoned immediately and the new infoContainer launched. Goto targets may be of the following types: chain, infoContainer, mid. Goto is forbidden to an infoContainer nested in another chain. Return values from objects which are targets of goto are lost, because there is nothing waiting on the returned value. A goto which targets this MID document is equivalent to a restart of this MID document. Spawn indicates that control flow splits. Spawned targets may be of the following types: infoContainer, chain, pane, conditionalPane, mid. Both parent and child compete for focus in the application display space. Spawn is forbidden to an infoContainer nested in another chain. Spawn is forbidden to a pane nested in a client area. Return values from spawned objects are lost, because	

	there is nothing waiting on the returned value. When a spawn is encountered, control stops in the calling script, the target is flowed until it reaches an idle state, then the caller continues until it reaches an idle state. --> <!ELEMENT (gosub | goto | spawn) - O (%locs;)*> <!ATTLIST (gosub | goto | spawn) HyTime NAME ilink HyNames CDATA "linkends target" anchrole CDATA "me target" target IDREF #REQUIRED >	
20	<!-- get Get expresses that the information at the source be collected, concatenated, and rendered at the point of the get. If space is specified, the members of a target aggregate will be delimited by a single space before the data is concatenated. If normalize is specified, leading and trailing white space is removed, and multiple contiguous spaces are converted into a single space. --> <!ELEMENT get - O (%locs;)*> <!ATTLIST get HyTime NAME ilink anchrole CDATA "me source #AGG" HyNames CDATA "linkends source" source IDREF #REQUIRED space (space | noSpace) space normalize (normalize | noNormalize) noNormalize >	• A 'get' element may point at (i.e., have as its content source) another get element, and so on in a chain of indirection to the final data.
21	<!-- relationship The relationship form conforms to the architecture for a HyTime ilink. It expresses an authored relationship between two identified objects. The application must provide its own element and attribute declarations for hyperlinking according to the HyTime standard. This pseudo-declaration is provided as a model for the HyTime ilink. The generic identifier of the relationship governs the relationship semantic. The traversal semantic of the relationship is governed by the traversal attribute. If traversal is set to be undefined, traversal decisions will be left up to the application. Attributes may be added to change traversal from hotspot marking (interrupt) to hotspot information by request only (polling). This would prevent hotspot clutter in an on-line index, for example. <!element relationship - O (title, (%locs;)*)> <!attlist relationship HyTime NAME ilink	• Application of relationship applies to hotspots in text and graphics. In the case of graphics, the identification of an object to link to is via a notation (i.e., names of objects/zones must be available through a notloc). • A relationship of traversal type 'gosub' to a script will be treated like a function call, except that the script is lexically contained only within the MID instance, not within the other end of the relationship, and not at its own location within the SGML instance (because its container might not have been flowed). • Note that **relationship** is an architectural form.

	MID NAME #FIXED relationship relationshipName #CDATA #FIXED id ID #IMPLIED anchrole CDATA #FIXED "antecedent #AGG consequent #AGG" linkends IDREFS #REQUIRED privTrav NAMES #IMPLIED extra NAMES #IMPLIED intra NAMES #IMPLIED endterms IDREFS #IMPLIED aggtrav NAMES agg traversal (gosub | spawn | goto | undefined) spawn > -->	
22	<!-- CONTROLS -->	
23	<!-- menu This element declares a named and labeled menu of menus and menu items. If disable is specified, the menu label will be visible but the menu will be inaccessible ("grayed out"). The menu will be rendered when its label is selected from a rendered parent menu or menubar. --> <!ELEMENT menu - O (label, (menu | button | buttonGroup)*) > <!ATTLIST menu id ID #IMPLIED enable (enable | disable) enable >	
24	<!-- setState This element indicates that the state of the target object should be modified according to the attributes and content specified. Possible targets: menubar, menu, button, buttonGroup. More complicated substitutions should use the functionality provided by conditionalWidget. The toggle attribute tells whether the target should be toggled on, toggled off, that the toggle should be removed, or that no change to the toggle should take place. The enable attribute tells whether the target should be enabled or disabled ("grayed out") or that no change should be made. The action attribute tells whether to modify the target, to remove the target from its position, or to reset the target to its initial settings. The content attribute tells how to treat the content of the setState element. The subelements may be inserted before the target, after the target, or replace the target entirely. Replacing items on the menubar with a buttonGroup is not allowed. --> <!ELEMENT setState - O (menu | button | buttonGroup)* >	#Note: There is an ambiguity here concerning the scope of setState on a menubar (in global or local pool). A setState may apply to all instances of the menu in every scope, or may be limited to the current scope.

	<!ATTLIST setState
	  target IDREFS #REQUIRED
	  toggle (toggleOn | toggleOff | removeToggle | noToggleChange)
	noToggleChange
	  enable (enable | disable | noEnableChange) noEnableChange
	  action (modify | remove | reset) modify
	  content (insertBefore | insertAfter | replace) replace
	>	

25	<!-- button
	  A button represents a user interface activation control. The script
	is run when the button is activated. If specified, the name of the
	button must be unique within a button group. If toggleOn is
	specified, the button is rendered with a "toggled on" representation.
	If toggleOff is specified, the button is rendered with a "toggled off"
	representation. If disable is specified, the button will be visible
	but it will be inaccessible ("grayed out"). -->
	
	<!ELEMENT button - O (label?, script) >
	<!ATTLIST button
	  id ID #IMPLIED
	  name NAME #IMPLIED
	  toggle (toggleOn | toggleOff | noToggle) noToggle
	  enable (enable | disable) enable
	>	

26	<!-- fillin
	  A fillin represents a fill-in-the-blank widget. The initial value
	of the variable provides the initial text. When noEcho is
	specified, the user's input is not echoed to the display (e.g., for
	entering passwords). -->
	
	<!ELEMENT fillin - O (label, variable) >
	<!ATTLIST fillin
	  id ID #IMPLIED
	  echo (echo | noEcho) echo
	>	

27	<!-- dynamicList
	  A dynamicList represents a widget which allows a user to assign a
	value to a variable. When encountered, the label is rendered to name
	the widget. The expr is evaluated; the results become the option
	list, and the option list is rendered. If notRestricted is specified,
	the user may enter a value which is not on the option list. The
	script gets run when the user makes a selection. -->
	
	<!ELEMENT dynamicList - O (variable, label?, expr, script) >
	<!ATTLIST dynamicList
	  type (pickOne | pickMany) pickOne
	  restricted (restricted | notRestricted) notRestricted
	>	

28	<!-- label A label is made up of any combination of retrieved text, the results of evaluation of expressions, parsed character data, and icons. It is rendered when its container is rendered, in such a way as to preserve the semantic of grouping.--> <!ELEMENT label - O (get | expr | #PCDATA | icon)* >	
29	<!-- DATA TYPES -->	

30	<!-- text
	  Groups text items. -->
	
	<!ELEMENT text - O (get | expr | #PCDATA | specialText | title | paragraph | orderedList | unorderedList)* >
	<!ATTLIST text
	  id ID #IMPLIED
	>	

31	<!-- specialTextTypes
	  Lists the types of text which are recognized as special. -->
	
	<!ENTITY % specialTextTypes
	"visualPunch | foreignWord | semanticStress | newTerm | bibliographicReference |
	wordAsWord | wordAsDefinition | informalName | properObject | mathExpression | acronymExpansion | anchor | none" >
	
	<!-- specialText
	  Indicates that the contained text is qualified by some semantic.
	-->
	
	<!ELEMENT specialText - O (get | expr | #PCDATA | specialText)* >
	<!ATTLIST specialText
	  id ID #IMPLIED
	  type (%specialTextTypes;) none
	>
		• specialText will be used for semantic indications. The type may be associated to a particular appearance in the browser by an external stylesheet.
		• The potential that there is a hotspot on a specialText is determined purely by the ilink (relationship) that points to it, and not by reason of its being specialText.

32	<!-- title Title indicates the title of the object which contains it. It is always to be rendered in such a way as to indicate that association. The contents of the title element are evaluated and concatenated in the order that they appear. --> <!ELEMENT title - O (get | expr | #PCDATA | specialText)* >	
33	<!-- paragraph Indicates the contained text is regarded and rendered as a paragraph. -->	

	<!ELEMENT paragraph - O (get | expr | #PCDATA | specialText)* > <!ATTLIST paragraph id ID #IMPLIED >	
34	<!-- orderedList, unorderedList These represent two types of list. An ordered list is typically rendered with ascending identifying numbers, letters, etcetera. An unordered list is typically rendered with bullets instead. Items in either kind of list must be rendered in the order they appear lexically. --> <!ELEMENT (orderedList | unorderedList) - O (title?, item+) > <!ATTLIST (orderedList | unorderedList) id ID #IMPLIED >	• orderList, unorderedList are not affected by the MID support attribute for list. The support option list applies to the use of only the function type and varible type “list”. See also element #9, page 13.
35	<!-- item Represents a list item. --> <!ELEMENT item - O (get | expr | #PCDATA | specialText | orderedList | unorderedList)* >	• orderList, unorderedList are not affected by the MID support attribute for list. The support option list applies to the use of only the function type and varible type “list”. See also element #9, page 13.
36	<!-- EXTERNAL NOTATIONS -->	
37	<!-- icon, graphic, audio, video, animation Access to external notations is made from these elements. --> <!ELEMENT (icon | graphic | audio | video | animation) - O (get | expr)* > <!ATTLIST (icon | graphic | audio | video | animation) id ID #IMPLIED >	
38	<!-- TABLE ELEMENT TYPES -->	
39	<!-- tableTypes The table entity is used to allow the application to substitute any table type into the pane declaration.--> <!ENTITY % tableTypes "fcsTable" >	
40	<!-- fcsTable The fcsTable conforms to the HyTime finite coordinate space. It expresses the abstract layout of a table without imposing assumptions about how the tabular information will be rendered or used. To implement fcs tables, applications must implement the following architectural forms as required by the HyTime standard: fcs, evsched, axis, event, extlist, measure, granule. The element declarations	• Note that fcsTable is a very general, and non-implementation specific way to represent data that is typically rendered in tables. The table type is in an overrideable entity in order to allow interim use of more conventional tables in the name of expediency.

	below are provided as examples that conform to the needed HyTime architectural forms; they will be recognized by conforming HyTime engines.
	
	The fcsTable element contains event schedules. Its axisdefs attribute lists the generic identifiers of the axes that make up the space. The event elements contained in evscheds are scheduled on each of the axes of the fcs. -->
	
	<!ELEMENT fcsTable - O (evsched+) >
	<!ATTLIST fcsTable
	  HyTime NAME fcs
	  MID NAME fcsTable
	  id ID #IMPLIED
	  axisdefs CDATA #FIXED "x y"
	>	

41	<!-- Each axis is declared with a specific measurement domain (here, virspace) and with a specified dimension.
	
	Specific axes with specific axis dimensions must be declared for each instantiation of table type. The fcsTable and the evsched refer to the generic identifier of the desired axis.
	
	NOTE: The axis dimensions "4" and "5" are example dimensions. Particular tables may have any dimension required, theoretically up to the the high quantum count value in HyTime. -->
	
	<!ELEMENT x - O EMPTY >
	<!ATTLIST x
	  HyTime NAME axis
	  MID NAME axis
	  axismeas CDATA #FIXED "virspace"
	  axisdim CDATA #FIXED "4"
	>
	
	<!ELEMENT y - O EMPTY >
	<!ATTLIST y
	  HyTime NAME axis
	  MID NAME axis
	  axismeas CDATA #FIXED "virspace"
	  axisdim CDATA #FIXED "5"
	>	

42	<!-- The axisord attribute of the evsched element type dictates the order in which dimension specifications are to be listed in the extent list extlist of every event: first the spec for x, then the spec for y. The #FIXED value of the basegran attribute of evsched establishes a base measurement unit for scheduling, in this case a "virtual space	

	unit" (vsu). -->

<!ELEMENT evsched - O (event)* >
<!ATTLIST evsched
  HyTime NAME evsched
  MID NAME evsched
  id ID #IMPLIED
  axisord CDATA #FIXED "x y"
  basegran CDATA #FIXED "vsu"
  gran2hmu NUMBERS #FIXED "1 1"
  overrun (error \| wrap \| trunc \| ignore) error
> | |
| 43 | <!-- event, extlist
  The event and extlist elements are used by fcs but do not require specific-case architectural form initialization. They may be used as they appear below. -->

<!ELEMENT event - O (get \| expr \| #PCDATA)* >
<!ATTLIST event
  HyTime NAME event
  id ID #IMPLIED
  exspec IDREFS #REQUIRED
>

<!ELEMENT extlist - O (#PCDATA) >
<!ATTLIST extlist
  HyTime NAME extlist
  id ID #IMPLIED
> | |
| 44 | <!-- measure, granule
  The evsched element form requires a measure definition to occur somewhere in the document. The following one is an minimal example.

<measure smu=VIRSPACE>
  <granule gn=vsu gd="1 1 VIRSPACE">
</measure>

-->

<!ELEMENT measure - O (granule+) >
<!ATTLIST measure
  HyTime NAME measure
  id ID #IMPLIED
  smu NAME #REQUIRED
>

<!ELEMENT granule - O EMPTY>
<!ATTLIST granule
  HyTime NAME granule | |

	gn CDATA #REQUIRED gd CDATA #REQUIRED >	
45	<!-- SEMANTIC GROUPING -->	
46	<!-- paneGroup Panes are grouped with some semantic intention of the author. They are rendered with a title when encountered.--> <!ELEMENT paneGroup - O (title?, (pane | paneGroup | conditionalPane)*) > <!ATTLIST paneGroup id ID #IMPLIED >	
47	<!-- widgetGroup A widgetGroup is an optionally labeled group of widgets. All of its contents are rendered when encountered. The optional script is run after rendering the widget group, in order that setState may be called to set the toggled wigets. --> <!ELEMENT widgetGroup - O (label?, (widgetGroup | conditionalWidget | buttonGroup | dynamicList | button | fillin)*, script?) >	
48	<!-- buttonGroup Represents a labeled group of buttons. The semantic of the grouping is expressed in the type attribute. An indication of pickOne means only one button in the group may be selected. An indication of pickMany means any number may be selected. If a default is specified, the named buttons are preselected. If no default is specified and the type is pickOne, the first button is preselected. --> <!ELEMENT buttonGroup - O (label?, button*) > <!ATTLIST buttonGroup id ID #IMPLIED type (pickOne | pickMany | button) button default NAMES #IMPLIED >	
49	<!-- menubar This element declares a menu bar as a collection of menus and buttons. The menubar will be rendered when an infoContainer that points to it via its menubar attribute is rendered.--> <!ELEMENT menubar - O (menu | button)+ > <!ATTLIST menubar id ID #IMPLIED >	

50	<!-- CONDITIONALS -->	
51	<!-- conditionalPane When a conditionalPane is encountered, the expression is evaluated. Each successive expression is evaluated until one is equal to the first. The paneGroup corresponding to the match is rendered. If no expressions match, the final paneGroup, if present, is rendered as a default. This construct is reevaluated when a reflow command is issued for this element or an element in the proper ancestry of this element. --> <!ELEMENT conditionalPane - O (expr, (expr, paneGroup)*, paneGroup?) > <!ATTLIST conditionalPane id ID #IMPLIED >	• Conditional panes are intended to be used for alternate displays of panes, as in alternate language presentation, rather than to implement preconditions such as those found in MIL-D-87269; a conditional pane is not intended to be the implementation of an if-step, nor is it intended to allow an entire interactive electronic technical manual to be implemented with a single infoContainer.
52	<!-- conditionalWidget When a conditionalWidget is encountered, the first contained expression is evaluated. Each successive contained expression is evaluated, in order, until one is equal to the first. The widgetGroup corresponding to the match is rendered. If no expressions match, the final contained widget group (if present) is rendered as a default. --> <!ELEMENT conditionalWidget - O (expr, (expr, widgetGroup)*, widgetGroup?) > <!ATTLIST conditionalWidget id ID #IMPLIED >	
53	<!-- reflow When a reflow is encountered, the entire subtree of the target of the reflow statement is rendered again. The target must be something in the current infoContainer, and all current states within the scope of the infoContainer will be respected. When no target is specified, the entire current infoContainer will be rendered again. If multiple targets are specified they are rendered again in the order given. --> <!ELEMENT reflow - O EMPTY > <!ATTLIST reflow target IDREFS #IMPLIED >	
54	<!-- SCRIPTING -->	
55	<!-- script Scripts are evaluated depending on their context. First the declarations are evaluated, then the statements. The return type for the script is specified using the functionType attribute. --> <!ELEMENT script - O ((vardecl | funcdecl | xenodecl)*, statements) >	• If you GOTO from a script, the flow is terminated, and statements logically following the GOTO will not be executed.

	<!ATTLIST script
	  id ID #IMPLIED
	  functionType (%functionTypes;) atom
	>
		#Note: There appears to be no need for functionType, because everything that catches a return from a script could catch the return from what contains the script (e.g., a pane has a functionType already).

56	<!-- name The name of a function, variable, xenodecl, or scriptLabel is created by evaluating the contained elements in the order they appear and concatenating the results. After the data is concatenated, leading and trailing whitespace characters are ignored, and multiple whitespace characters are replaced by a single space. SGML NAME characters are folded according to the NAMECASE GENERAL parameter of the governing SGML declaration. --> <!ELEMENT name O O (get | expr | #PCDATA)* >	• Name should resolve to #PCDATA (preferred) or strings. Developers will expect 'get' to return only these types. The application should be prepared to handle variable, argument, scriptLabel, and xeno function names in the character set of the MID document • Empty names are not particularly desirable. #Note: The range of character sets in a MID document should be defined to guard against implementation specific requirements driven by differing character sets.
57	<!-- statements Statements are evaluated as directed by the context, in the order they appear. NOTE: Although the absence of this container would not create ambiguities in the MID language (i.e., this container is redundant), it is provided as a convenience to MID script interpreters. --> <!ELEMENT statements O O (expr | if | loop | break | switch | jump | scriptLabel | goto | spawn | return | reflow | messageArea | setState)* >	
58	<!-- DECLARATIONS -->	

59	<!-- Declarations
	  Declarations are processed and bound to the declared names in the order the declarations are encountered. If a variable declaration initializer contains a reference to another variable, the other variable must have been declared and initialized prior to the referring declaration. -->
	
	<!-- vardecl
	  The vardecl element binds a name to a run-time storage location. Variables must be declared before use. The expr initializes the variable. The variableType attribute specifies the type of the variable. Every variable type has a default initialization: zero for integer and float types, false for boolean, and null for list and string. The default sgmlchar is zero. A local name which is the same as a name declared higher in the scope stack renders the higher named object unreferenceable (the local name is said to "shadow" the higher one). -->	

	<!ELEMENT vardecl - O (name, expr?) > <!ATTLIST vardecl variableType (%variableTypes;) string >	

60	<!-- funcdecl
	  The funcdecl element binds a function name with argument list, local state and statement list. Vardecl names shadow argdecl names. The number of arguments, their types, and their order are always fixed. The functionType attribute specifies the return type of the function.
	-->
	
	<!ELEMENT funcdecl - O (name, argdecl*, vardecl*, statements) >
	<!ATTLIST funcdecl
	  functionType (%functionTypes;) atom
	>	

61	<!-- argdecl
	  The argdecl element binds an argument name to a passed value. Arguments to functions are passed by value. -->
	
	<!ELEMENT argdecl - O (name) >
	<!ATTLIST argdecl
	  variableType (%variableTypes;) string
	>
		• Note that the MID DTD does not support default values of arguments. The function call should contain exactly as many arguments as the funcdecl specifies, or the MID script is in error.

62	<!-- STRING and LIST OPERATIONS -->	
63	<!-- stringOperations The operations specific to string manipulation are collected here. --> <!ENTITY % stringOperations "strlen | substr | strcat | fold | isnull | isstring" >	#Note: A new function, CharAt(string, pos), is under consideration, which returns a (sgml)char in a specified position from a string. This can be accomplished with my proposed casting by doing substr and then casting the result to a char.
64	<!-- listOperations The operations specific to list manipulation are collected here. --> <!ENTITY % listOperations "list | cons | car | cdr | append | isnull | islist | nth | count" >	#Note: As noted under cons element below, this function can be performed as a special case of append.
65	<!-- expr The expr element is evaluated as one of its contained elements. A copy of the result is returned. NOTE: Although the absence of this container would not create ambiguities in the MID language (i.e., this container is redundant), it is provided as a convenience to MID script interpreters. -->	• Casting recommendations are shown in the primitives entity declaration.

	<!ELEMENT expr - O (assign | variable | constant | function | add | multiply | subtract | divide | modulus | eq | lt | gt | le | ge | and | or | ne | not | gettype | %stringOperations; | %listOperations; | gosub) >	
66	<!-- variable The contents of the storage bound to the name are returned to the caller. --> <!ELEMENT variable - O (name) >	• Variable will be searched in nearest lexically enclosing scope. For example, if the reference is inside of a function, but the variable is not within the function, then the scope of the pane containing the function will be tried next, then the infoContainer containing the pane, then the MID itself (ie, the declarations directly in the mid element).
67	<!-- assign The expression is evaluated and the results are placed in the variable storage bound to the name. --> <!ELEMENT assign - O (name, expr) >	• Implicit casting will be performed as necessary by the script interpreter.
68	<!-- function The arguments are evaluated in the order in which they appear and the results are passed as arguments to the named funcdecl or xenodecl. --> <!ELEMENT function - O (name, argument*) >	• Default arguments are not supported; the function call should have the same number of arguments as the function declaration had. • Argument types should match declarations, or should be designed in accordance with known casting rules (coercible)
69	<!-- argument The expression is evaluated as the argument passed to a function. --> <!ELEMENT argument - O (expr) >	
70	<!-- add, multiply, subtract, divide, modulus, eq, lt, gt, le, ge, and, or, ne, not The expressions are evaluated and the operation is applied according to the data type. --> <!ELEMENT (add | multiply) - O (expr+) > <!ELEMENT (subtract | divide | modulus) - O (expr, expr) > <!ELEMENT (eq | lt | gt | le | ge | and | or) - O (expr)+ > <!ELEMENT ne - O (expr, expr) > <!ELEMENT not - O (expr) >	• Numeric operations (add, mult, sub, div, mod) should have numeric arguments (int32, uint32, int64, uint64, float32, float64). • Lexicographical relations (lt, gt, le, ge) may be used on two numeric args, two sgmlchar args, or two string args. SGMLChar can be compared to any string whose length is one. • Boolean operations (and, or, not) should have boolean arguments (boolean).

		• Equality operators (eq, ne) can have arguments in several categories (including chars, strings, lists, numeric, boolean). Strings and lists should only be compared within their own type. Numeric/boolean types can be compared (casting rules will apply). SGMLChar can be compared to any string whose length is one. • The application is expected to implement all arithmetic results, including implementation dependent ones, so that (eq a b) will be true if and only if (ne a b) is false, for any a and b. • Lexicographical comparison of chars or strings needs to be standardized. For now, assume that a char may be compared lexicographically with a string, with the proviso that (e.g.) 'f' precedes 'fa'. • Similarly, a char may be compared for equality with a string of length greater than one - it will be unequal. #Note: There is a difficulty if one string (or char) is from a different notation than the other. Then the results are implementation dependent. #Note: Addition of BitwiseAnd and BitwiseOr operators (or equivalents with better names), whose arguments should be integers, is under consideration.
71	<!-- constant The contents are evaluated and a value of the given constant type is constructed. The contantType attribute specifies the data type. The regular expressions and semantics for the MID primitives are as follows, with keywords and letters folding case by the rule of the SGML namecase. boolean ("true"|"false"|"1"|"0"|"yes"|"no") "true", 1, and "yes" are equivalent; "false", 0, and "no" are equivalent. We declare the following as shorthand: DIGIT = ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9") NZDIGIT = ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9") int32, int64 ("+"|"-")?, NZDIGIT, DIGIT* These constants are base 10 representation only. "+" indicates positive; "-" indicates negative. uint32, uint64 NZDIGIT, DIGIT*	• Strings or lists with nothing in them are null. • Empty constants of type number or boolean are discouraged, and might not be handled consistently by software applications. • 'get' as an element of a constant should point to #PCDATA, or a notation. • If get as an element of a constant points to a function call, it should be a call to a xeno function, and the MID author should be indifferent as to whether the application evaluates the constant by calling the xeno function only the first time, or every time. • Be sure to fold with normalize 'on' before using strlen, substr, or isnull if you want to get a length or sub without the CR, LF, RS, RE, etc. Otherwise these will include the CR, etc in the string.

	These constants are base 10 representation only. float32, float64 ("+"|"-")?, DIGIT+, ".", DIGIT+ , ("e", ("+"|"-")?, NZDIGIT, DIGIT*)? The "e" means "times-ten-to-the". Digits are required on both sides of the decimal point. The mantissa is represented in 16 bits for float32, and 32 bits for float64. The remaining bits are reserved for the exponent. sgmlchar And any single valid sgml character string any valid string of sgml characters NOTE: In string and sgmlchar, record start (RS) and record end (RE) are ignored according to the rules in ISO 8879. --> <!ELEMENT constant - O (get | #PCDATA)* > <!ATTLIST constant constantType (%constantTypes;) #REQUIRED normalize (normalize | noNormalize) noNormalize recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter >	
72	<!-- if, else If the expression evaluates to true, the statements in the statements element are executed. Otherwise, the statements within the else are evaluated. --> <!ELEMENT if - O (expr, statements, else?) > <!ELEMENT else - O (statements) >	
73	<!-- loop The expression is evaluated. If the expression is true, the statements are executed. The expression is reevaluated and the statements are re-executed until the expression returns false. --> <!ELEMENT loop - O (expr, statements) >	• The expr in 'if' or 'loop' should be boolean or numeric. Casting applies in the numeric case. • Jumps in and out of loops are prohibited. The stack will be busted if you try this, thereby causing much weaping and finger pointing by the programming staff.
74	<!-- continue, break Continue causes execution to resume at the top of the nearest enclosing loop, whereupon the loop's expr is reevaluated. As always, if the expr evaluates to false, execution resumes at the statement following the loop. Break causes execution to resume at the statement	

	following the nearest enclosing loop or switch. If there is no lexically enclosing loop, continue is ignored. If there is no lexically enclosing loop or switch, break is ignored. No stacks are affected. Jumping to a point within a loop initiates looping behavior. Jumping to a point within a switch causes the switch's expr to be evaluated automatically prior to execution of any statements. --> <!ELEMENT (continue | break) - O EMPTY >	
75	<!-- switch, case, default The expression is evaluated. Each of the expressions of the cases is evaluated in the order in which the cases appear until one matches the switch expression. If a match is found, the statements under the matched case are executed until a break is encountered. Control continues to the statements under the next case if no break is encountered; the interceding expr is not evaluated. If no case expression is matched, the default statements are executed.--> <!ELEMENT switch - O (expr, case*, default?) > <!ELEMENT case - O (expr, statements?) > <!ELEMENT default - O (statements?) >	• See rules for 'eq' in regard to expression comparison.
76	<!-- jump, scriptLabel Control immediately jumps to the named script label. Certain restrictions apply: script labels are scoped local to a given script. --> <!ELEMENT jump - O (name) > <!ELEMENT scriptLabel - O (name) >	
77	<!-- gettype Returns the type of the expression. Return type: string. --> <!ELEMENT gettype - O (expr) >	• The string will be exactly as listed in the original primitives and variableTypes entity declarations.
78	<!-- STRING OPERATIONS -->	
79	<!-- strlen Returns the length of the string. Return type: uint32. Returns zero as a 'reportable MID error' if expression is not a string. --> <!ELEMENT strlen - O (expr) >	
80	<!-- substr First expr: string. Second expr: start position. Third expr: run length. This construct returns the substring of the string, given start position and run length. Start position is counted from 1.	• First expr must be a string. Second and third expr must be numeric.

	Unspecified run length or run length greater than the length of the string indicates the rest of string. Returns null string if start position exceeds string length. Return type: string. --> <!ELEMENT substr - O (expr, expr, expr?) >	
81	<!-- strcat Returns the concatenation of the strings. Expressions in this element's immediate content which evaluate to non-strings are ignored. If space is specified, a space character is inserted between expressions. If normalize is specified, leading and trailing white space is removed, and multiple contiguous spaces are converted into a single space. Return type: string. --> <!ELEMENT strcat - O (expr)+ > <!ATTLIST strcat space (space | noSpace) noSpace normalize (normalize | noNormalize) noNormalize recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter >	
82	<!-- fold Returns the folded version of the string. Converts string to uppercase using SGML name case folding rules. The name attribute tells whether the name characters are to be folded according the SGML declaration rules for entity names or for general names. Return type: string. --> <!ELEMENT fold - O (expr) > <!ATTLIST fold name (general | entity) general normalize (normalize | noNormalize) noNormalize recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter >	
83	<!-- isnull This function returns true if the expression is a null (empty) string or a null list. It returns false otherwise. Return type: boolean. --> <!ELEMENT isnull - O (expr) >	
84	<!-- isstring Returns whether the expression is a string. Return type: boolean. --> <!ELEMENT isstring - O (expr) >	

85	<!-- LIST OPERATIONS -->	
86	<!-- list Each expression in the content of this element is evaluated and becomes an top-level item on the returned list. If no expressions are specified, this element returns a null list. A list in MID has the same binary tree implementation as lists in Lisp or Prolog. Return type: list. -->\ <!ELEMENT list - O (expr)* >	• Each expression creates a single element (which could be another list).
87	<!-- cons This function returns a list in which the result of evaluating the first expression is prepended to the list found in the second expression. The second expression must be a list. Return type: list. NOTE: The names cons, car, and cdr, while perhaps non-intuitive in English, is precisely meaningful in LISP; they were chosen deliberately to enhance interdisciplinary communications. -->\ <!ELEMENT cons - O (expr, expr) >	#Note: cons is somewhat redundant and may be removed from DTD at a future date. Because we only allow cons when the second element is a list, (cons (a) (b)) can be accomplished as (append (list (a)) (b)) where in both cases b is constrained to be a list.
88	<!-- car This function returns the car of the list, i.e, the first item of a cons pair. The expression must be a list. Return type: any. -->\ <!ELEMENT car - O (expr) >	
89	<!-- cdr This function returns all but the first item in a list (the second half of a cons pair). Return type: list. -->\ <!ELEMENT cdr - O (expr) >	
90	<!-- append This function returns the results of appending a list to a list. Return type: list. -->\ <!ELEMENT append - O (expr, expr) >	• Both expressions must be lists. The second expr is appended to the first. #Note: A generalized append could be specified with a content model of (expr, expr+).
91	<!-- islist This function returns true if the expression is of type list. Return type: boolean. -->\ <!ELEMENT islist - O (expr) >	
92	<!-- nth	

	expr1: list. expr2: integer. Returns the nth item of the list, counting from one, without recurring into nested lists. Returns the null list if there is no such item. Return type: atom or list. --> <!ELEMENT nth - O (expr, expr) >	
93	<!-- count Returns the number of items in the given list, without recurring into nested lists. Returns zero if expression is a list which has no members. Return type: uint32. --> <!ELEMENT count - O (expr) >	
94	<!-- EXTERNAL PROCESS -->	
95	<!-- xenodecl The xenodecl element binds a name and argument declarations to a call to an external notation. The functionType attribute specifies the return type of this construct.--> <!ELEMENT xenodecl - O (name, argdecl*, xeno) > <!ATTLIST xenodecl functionType (%functionTypes;) atom >	
96	<!-- xeno The xeno element represents a subclass of the HyTime notloc. Argument names from the containing xenodecl indicate substitution into the RCDATA of the xeno when the argument name is surrounded by the string tokens specified in argBegin and argEnd.--> <!ELEMENT xeno - O RCDATA > <!ATTLIST xeno HyTime NAME notloc id ID #IMPLIED qdomain IDREFS #IMPLIED qcontext IDREF #IMPLIED ordering (ordered | noorder) noorder set (set | notset) notset aggloc (aggloc | agglink | nagg) nagg argBegin CDATA "$(" argEnd CDATA ")" >	• Xeno functions are intended as a catch-all for functions that are not included in the MID. The referenced notation defines the standard way that the xeno is to be implemented in practice. In the near term, this may require that authors make assumptions about platforms, and deliver code (DLLs) to execute their functions in order to handle any returns needed. #Note: Support names may be needed for specific formats of graphics, text, audio, video, etc. These indicate a requirement for MIDReaders where there are well-known notations that do not need a specific author-defined xeno to execute. Use public notations where available, and name them in support declaration. Examples: BMP, JPEG, CGM, TIFF, WMF. #Note: A standard notation for DLLs may be useful. Define in terms of a string that can be passed via the RCData in the xeno: DLL Name Function entry point name Number of arguments Argument types and names

		Arguments in specific order Return type Calling convention

97	<!-- return
	  This element terminates processing of the nearest containing construct specified by the construct attribute, and returns the value resulting from evaluating the expression. If there is no expression, the return value is the default initialization for the stated return type. -->\n\n<!ELEMENT return - O (expr?) >
	<!ATTLIST return
	  construct (mid | chain | infoContainer | pane | alert | script | function) function
	>
		• Care should be exercised to insure that a return references a lexically enclosing construct. For example, an infocontainer may not call a global level function and have the global level function end the infocontainer with a <return construct=infocontainer>
		• A <return contruct=script> when executed within a function is equivelant to a <return contruct=function>. If a return with construct=script is executed outside a function, its result will be to terminate the script.

98	<!-- HYTIME -->	

99	<!-- security
	  Security is an implementation of the HyTime activity form. The security attribute tells what level of security. Elements mid and pane may point to a security element, thereby indicating the security level. The contained script (if any) will be run when the indicated activity (in this case, access) occurs.-->\n\n<!ELEMENT security - O (script)? >
	<!ATTLIST security
	  id ID #IMPLIED
	  HyTime NAME activity
	  actypes NAMES access
	  level (unclassified | confidential | secret | topSecret) unclassified
	>	

100	<!-- The following HyTime location types are instantiated directly from the HyTime standard. -->	

101	<!ELEMENT nameloc - O (nmlist | HyQ)* >
	<!ATTLIST nameloc
	  HyTime NAME nameloc
	  id ID #REQUIRED
	  ordering (ordered | noorder) noorder
	  set (set | notset) notset
	  aggloc (aggloc | agglink | nagg) nagg
	>	

102	<!ELEMENT nmlist - O (#PCDATA) > <!ATTLIST nmlist	

	HyTime NAME nmlist nametype (entity | element | unified) #REQUIRED obnames (obnames | nobnames) #REQUIRED docorsub ENTITY #IMPLIED dtdorlpd NAMES #IMPLIED >	
103	<!ELEMENT HyQ - O (#PCDATA) > <!ATTLIST HyQ HyTime NAME nmquery qdomain IDREFS #IMPLIED qcontext IDREF #IMPLIED notation NAME #FIXED HyQ delims CDATA #IMPLIED fn NAME #IMPLIED usefn NAME #CONREF args IDREFS #IMPLIED qpnpsn NAMES #IMPLIED qltnlmgi NAMES #IMPLIED >	
104	<!ELEMENT treeloc - O (marklist*) > <!ATTLIST treeloc HyTime NAME treeloc id ID #REQUIRED overrun (error | wrap | trunc | ignore) error treecom (treecom | ntreecom) ntreecom locsrc IDREFS #IMPLIED ordering (ordered | noorder) noorder set (set | notset) notset aggloc (aggloc | agglink | nagg) nagg >	
105	<!ELEMENT relloc - O (dimlist*) > <!ATTLIST relloc HyTime NAME relloc id ID #REQUIRED root IDREFS #IMPLIED relation (anc | esib | ysib | des | parent | children) parent overrun (error | wrap | trunc | ignore) error locsrc IDREFS #IMPLIED ordering (ordered | noorder) noorder set (set | notset) notset aggloc (aggloc | agglink | nagg) nagg >	
106	<!ELEMENT dataloc - O (dimlist*) > <!ATTLIST dataloc HyTime NAME dataloc id ID #REQUIRED	

	quantum (str | norm | word | name | sint | date | time | utc) str catsrc (catsrc | nocatsrc) nocatsrc catres (catres | nocatres) nocatres overrun (error | wrap | trunc | ignore) error locsrc IDREFS #IMPLIED ordering (ordered | noorder) noorder set (set | notset) notset aggloc (aggloc | agglink | nagg) nagg >	
107	<!ELEMENT marklist O O (marklist | #PCDATA)* > <!ATTLIST marklist HyTime NAME marklist >	
108	<!ELEMENT dimlist O O (dimlist | marklist | #PCDATA)* > <!ATTLIST dimlist HyTime NAME dimlist >	
109	<!ELEMENT proploc - O (qpn | #PCDATA) > <!ATTLIST proploc HyTime NAME proploc id ID #REQUIRED joint (joint | several) several apropsrc (apropsrc | solesrc) solesrc notprop (error | empty | ignore) ignore locsrc IDREFS #IMPLIED ordering (ordered | noorder) noorder set (set | notset) notset aggloc (aggloc | agglink | nagg) nagg >	
110	<!ELEMENT qpn - O (pn, spec?)+ > <!ATTLIST qpn HyTime NAME qpn id ID #REQUIRED >	
111	<!ELEMENT pn - O RCDATA > <!ATTLIST pn HyTime NAME pn >	
112	<!ELEMENT spec - O ((qpn | qltn)+ | pval) > <!ATTLIST spec HyTime NAME spec >	

113	<!ELEMENT qltn - O RCDATA> <!ATTLIST qltn HyTime NAME qltn >	
114	<!ELEMENT pval - O RCDATA > <!ATTLIST pval HyTime NAME pval >	

115	<!ELEMENT notloc - O ANY >
	<!ATTLIST notloc
	  HyTime NAME notloc
	  id ID #REQUIRED
	  qdomain IDREFS #IMPLIED
	  qcontext IDREF #IMPLIED
	  fn NAME #IMPLIED
	  usefn NAME #CONREF
	  args CDATA #IMPLIED
	  ordering (ordered | noorder) noorder
	  set (set | notset) notset
	  aggloc (aggloc | agglink | nagg) nagg
	>	

116	<!ELEMENT bibloc - O ANY >
	<!ATTLIST bibloc
	  HyTime NAME bibloc
	  id ID #REQUIRED
	  qdomain IDREFS #IMPLIED
	  qcontext IDREF #IMPLIED
	  fn NAME #IMPLIED
	  usefn NAME #CONREF
	  args CDATA #IMPLIED
	>	

7. Alphabetical index of elements

8. Summary of changes since original release of MID

This is a condensed and summarized list of changes that have been implemented in the MID DTD as a result of the 1995 design and development efforts.

Containers

infoContainer

- **menubar** collected by IDREF
- Added local **pool** for variables scoped to the infoContainer
- Has return type

endic

- Removed **endic** element that used to indicate when to close an **infoContainer**. Now we have a generalized **return** element for use in scripting, with a type that indicates which construct is ending. This removes an artifical distinction between return and endic, and requires fewer tags for the same functionality.

chain

- Added **chain** to allow author to specify a sequence of **infoContainers** that should not be entered in the middle. This is to be used e.g., in a procedure where users should proceed in order through a series of procedural steps for safety or other reasons.

popupDialog, footerbar

- Consolidated into **pane**, as these are just special cases of panes.

pane

- Removed 1,2,3,4 for **pane** numbering - **paneGroup** now allows more general specification of how panes should be semantically grouped, rather than how they should be ordered.
- Added function type and id to attribute list.

alert

- Moved "alerttype" to attribute.
- Removed hotspot restriction.
- added alert to clientarea element

appGlobals

- Application globals became main **pool** in <**mid**> element.

clientArea

- Changed to allow more than 4 **panes**, which can be grouped. See footer bar comment below for how to make footer bar. **messageArea** is retained and can be set in a script. The original script was really for "process panes" which are now handled through **xeno** in an **infoContainer**

footerBar

- Deleted footerBar element. Footer bar can be rendered as a **pane** with a buttongroup.

messageArea

- Added "expr"

poolContainer

- Deleted; replaced by the "pool" element which allows for additional reusable components.

popupDialog

- Deleted; now handled by pane with transition type

Transitions & Links

relationship

- Added example architechtural form for elements that are to be used for hotspots and hypermedia webs across documents.

hotspot

- Deleted; now handled by relationship

grotspot

- Graphic hotspot element was deleted. This was a temporary element used in MID Phase 1 examples. The function performed by the grotspot can be accomplished using the relationship element.

spawn

- This is a new element type that allows a script to launch an infoContainer whose scope is maintained simultaneously with the present scope.

Controls

PickOne and pickMany

- merged into dynamicList with attribute to represent pickOne or pickMany
- pickmany and pickone (deleted. became attribute on buttongroup)
- put info on the button

button

- labels can now contain icons instead of graphics
- defaults in “buttongroup” can be used to define a “default” button

dynamiclist

- element added to allow for assignment of value to a varible based on selection of one or more items from a list

fillin

- added required label. Attribute for echo of user input was added.

is default

- removed from DTD, replaced by “default” attribute for buttongroup element

label

- allow use of icon for label
- reduced scripting capability

setState

- can now adjust menus on the fly

menu

- added attrbiutes id and enable

menu item

- removed become "button" on "menubar"

menu text

- removed became "label" on "button" on "menubar

prompt

- removed

user interaction

- removed became widgetgroup

Data Types

specialText

- added special text
- emphasis (deleted. became specialtext)
- redefined as necessary text or special types

default text

- removed

fcsTable

- added fcs table types, commented example arch form

table types

- added as a overridable entity for allowed table types. Allows for any table type such as CALS

Removed explicit graphic elements

Removed explicit CALS table

animation

- attribute for id added

array

- removed, replaced by list element

graphic primitive

- removed

paragraph

- changed. "emphasis" is now "specialtext", lists have been added.

title bar

deleted. became title

Semantic Grouping

paneGroup

- added to allow for grouping of like panes such as in a procedure

widgetGroup

- widget group (new) replaces userinteraction

Conditionals

Added the following new elements:

- conditionalPane
- conditionalWidget
- reflow (for conditionals)

Scripting

Scripts

- has return type

name (changed. end-tagging, added "expr")

variables

- moved types into attribute values
- variable declaration (changed. arrays became lists in expression. type is in attribute)

string

- added string expressions
- is null (new)

list

- added list expressions

(get | script | #PCDATA)*

- changed to (get|expr|#PCDATA)*

Defined scopes more carefully

argument declarations (changed. moved type to attribute)

assign (essentially unchanged. variable became "name")

constant (changed. added "get" ability and "type" in attribute)

expression (changed. abbreviated to expr. string and list operations added via entity)

fold (new)

funcdecl (changed. "type" moved to attribute list)

gettype (new. returns "type")

not equal (changed. locked down to two (2) "expr")

return (changed. attribute list added)

- Added type attribute to <return >; the type tells which construct is ending, be it a function, script, pane, infoContainer, chain, or mid.

script (changed. attribute list added)

External Processes

- graphic (changed to external process)
- icon (changed. behaves now like graphic)
- xeno func (deleted. became a call to a function declared by a "xenodecl")
- xenofunc (removed <xenofunc > as a call to a xenodecl. <function > may be used to call both funcdecls and xenodecls.

HyTime Location & Linking

removed locContainer

location container (deleted. became "pool")

HyTime & SGML Management

security

- added security as HyTime activity to the mid and pane

Support

- invented support keywords

Structural changes:

- Rearranged grouping of elements
- Set SGML NAMECASE GENERAL NO
- Enforced mixedCapitals
- Defined parameter entities %locs, %primitives, %functionTypes, %specialTextTypes
- Attribute "type **%functiontypes**;..." replaced " **type**" element
- Element "**specialText" with associated attribute "specialTextTypes"** replaces element " **emphasis"** as a more general case

documentation

- cleaned and honed documentation
- regularized expression of comments
- the MID DTD stands more on its own

mid element

- collapsed various buckets into single pool
- has return type

Appendices

A. Processable MID DTD

```
<!SGML "ISO8879:1986"
   CHARSET BASESET "ISO 646-1983//CHARSET International
   Reference Version
   (IRV)//ESC 2/5 4/0" --2/5 was 2/8--
   DESCSET    0   9    UNUSED
      9    2    9
      11   2    UNUSED
      13   1    13
      14   18   UNUSED
      32   95   32
      127  1    UNUSED
   BASESET "ISO Registration Number 100//CHARSET ECMA-94
      Right Part of Latin Alphabet Nr. 1//ESC 2/13  4/1"
   DESCSET    128  32    UNUSED
      160   5    32
      165   1    UNUSED
      166   8    38
      254   1    127
      255   1    UNUSED
   CAPACITY    SGMLREF
   TOTALCAP    175000
   GRPCAP    70000
   ATTCAP    50000
   SCOPE    DOCUMENT

SYNTAX
   SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
    18 19 20 21 22 23 24 25 26 27 28 29 30 31 127 255
BASESET "ISO 646-1983//CHARSET International Reference Version
(IRV)//ESC 2/5 4/0"
   DESCSET    0   128  0
   FUNCTION   RE          13
      RS          10
      SPACE       32
      TAB  SEPCHAR    9
   NAMING    LCNMSTRT    ""
      UCNMSTRT    ""
      LCNMCHAR    "-."
      UCNMCHAR    "-."
      NAMECASE    GENERAL    YES
      ENTITY NO
   DELIM    GENERAL    SGMLREF
      SHORTREF NONE  --short references disabled for time being--
   NAMES    SGMLREF
   QUANTITY    SGMLREF    LITLEN    2048
      NAMELEN    32
      ATTCNT    80
      GRPCNT    80  --used default value of 32 before--
FEATURES
MINIMIZE    DATATAG    NO    OMITTAG    YES    RANK    NO
SHORTTAG    YES  -- SHORTTAG NO no CALS SGML Declaration.
Considered desirable
            to minimize MID instances.  --
LINK    SIMPLE    NO    IMPLICIT    NO    EXPLICIT    NO
OTHER    CONCUR    NO    SUBDOC    NO    FORMAL    YES
APPINFO    "HyTime"
>
<?HyTime VERSION "ISO/IEC 10744:1992" HYQCNT=32>
<?HyTime MODULE base exidrefs>
<?HyTime MODULE measure>
<?HyTime MODULE locs anydtd coordloc HyQ multloc query relloc>
<?HyTime MODULE links manyanch>
```

```
<!--
        MID: Metafile for Interactive Documents
             Document Type Definition

This document type definition shall be identified by the following
declaration:
    PUBLIC "-//MID//DTD MID Document Type Definition 19951201//EN"
-->

<!-- NOTATIONS -->
<!NOTATION SGML PUBLIC
"+//ISO 8879:1986//NOTATION Standard Generalized Markup
Language//EN" >

<!NOTATION HyTime PUBLIC
"+//ISO/IEC 10744:1992//NOTATION Hypermedia/Time-based
Structuring Language//EN" >

<!NOTATION HyQ PUBLIC
"+//ISO/IEC 10744:1992//NOTATION HyTime Query Notation//EN" >

<!NOTATION virspace PUBLIC
"+//ISO/IEC 10744:1992//NOTATION Virtual Measurement Unit//EN" >


<!-- ENTITIES -->
<!-- locs
  These are the HyTime Location addresses used by the MID.  -->

<!ENTITY % locs "nameloc | treeloc | dataloc | notloc | proploc | relloc |
bibloc"
>

<!-- primitives, functionType, variableType, constantType
  The MID script primitives are listed in the entities below. The
application may choose to override these declarations to extend or
constrain the MID definition.  An atom is string or any one of the
primitives. -->

<!ENTITY % primitives "boolean | int32 | uint32 | int64 | uint64 | float32 |
float64 | sgmlchar" >
<!ENTITY % functionTypes "%primitives | string | atom | list | any | void"
>
<!ENTITY % variableTypes "%primitives | string | atom | list | any" >
<!ENTITY % constantTypes "%primitives | string" >

<!-- MID -->
<!-- mid
```

The mid element is the document element for a mid application. The vardecls, funcdecls, and xenodecls in its immediate content are global declarations to the MID application. The MID instance is processed by first processing the global declarations and then the master script. The MID returns the results of evaluating its master script. The type of the resulting data is given as the value of the functionType attribute. This specification is redundant and is made solely for convenience. It is a reportable MID error (RME) if the type of the return value of the master script does not match the return value of the MID.

Date and version hold human-readable strings for specifying the date and version of this document.

The docmdu attribute specifies the measurement domains of the document's finite coordinate spaces (fcs) and the least common unit for computing dimensions in each fcs.

The security attribute identifies the security designation for this MID document. Security is implemented as a HyTime activity policy.

The following are support options for the support statement. The names may be listed in any order.

conditionalPane
  This document may contain conditional panes.

conditionalWidget
  This document may contain conditional widgets.

fcsTable
  This document may contain MID fcs tables.

list
  If list is specified, this document may use the list data structure and the list expressions. If list is not specified, list must be deleted from the functionTypes and variableTypes parameter entities.

MIL-M-87268
  This document is intended to be used in connection with software whose user interface strictly conforms to MIL-M-87268.

nonMID
  This document may contain addresses of locations in external SGML/HyTime documents which are not MID documents.

query

```
  This document may contain queries which address locations in
external SGML/HyTime documents which are not MID documents.  If
query
support is specified, nonMID support is implied.

relationship
  This document may contain MID relationship forms.

 spawn
  If spawn is specified, this document may contain spawn elements.

string
  If string is specified, this document may use the string data
structure and the string expressions.  If string is not specified,
string must be deleted from the functionTypes and variableTypes
parameter entities.

xeno
  If xeno is specified, this mid document may use the xenodecl and
xeno elements.  All data content notations must be declared using
notation declarations in the DTD.

-->

<!ELEMENT mid - O (( vardecl | funcdecl | xenodecl)*, script, pool?) >
<!ATTLIST mid
  HyTime NAME #FIXED HyDoc
  id ID #IMPLIED
  functionType  (%functionTypes;) atom
  date CDATA #IMPLIED
  version CDATA #IMPLIED
  docmdu CDATA #FIXED "virspace 1 1"
  HyNames CDATA "activity security"
  security IDREFS #IMPLIED
  support NAMES "conditionalPane conditionalWidget fcsTable list
    MIL-M-87268 nonMID query relationship spawn string xeno"
>

<!-- CONTAINERS -->
<!-- infoContainer
  When an infoContainer is accessed, its declarations are processed.
The menubar is built from the list of menubars given in the attribute
value, then the script ( which may contain adjustments to the menubar
in setState commands) is executed.  After this, the title, alert, and
clientArea are processed in the order they appear.  The functionType
attribute specifies the return type of this construct.-->

<!ELEMENT infoContainer - O (( vardecl | funcdecl | xenodecl)*, script?,
title?,
alert*, clientArea, pool?) >
<!ATTLIST infoContainer
  id ID #IMPLIED
  menubar IDREFS #IMPLIED
  functionType (%functionTypes;) atom
>

<!-- chain
  Access to infoContainers within a chain is restricted to
infoContainers within that chain.  When a chain is accessed, its first
contained infoContainer is processed.  -->

<!ELEMENT chain - O ( infoContainer)* >
<!ATTLIST chain
  id ID #IMPLIED
>


<!-- tableTypes
  The table entity is used to allow the application to substitute any
table type into the pane declaration.-->

<!ENTITY % tableTypes "fcsTable" >


<!-- pane
  A pane is a single user interface presentation, which is rendered
when it is encountered.  A pane encapsulates a scope.  A get within a
pane causes the target to be rendered on this pane.  Scripts within
panes are run when the pane is rendered.  The return value of the script
is the return value of the pane.  It is a RME if the type of the pane
and the containing script are not the same.  A pane is modeless when
contained in a client area or when called from spawn.  A pane is modal
when called from gosub.

The security attribute identifies the security designation for this
pane.  Security is implemented as a HyTime activity policy.

-->

<!ELEMENT pane - O (( vardecl | funcdecl | xenodecl)*, title?,
( text | %tableTypes; | graphic | audio | video | animation | widgetGroup |
get |
script)) >
<!ATTLIST pane
  id ID #IMPLIED
```

```
 functionType (%functionTypes;) atom
 HyNames CDATA "activity security"
 security IDREFS #IMPLIED
>

<!-- alert
 An alert represents a modal popup window with the contained
information.  The contents of the alert are evaluated and rendered in
the order they are encountered.  The alert is popped down when a
return alert statement is encountered in the button script.  The type
attribute indicates the semantic of the alert.  -->

<!ELEMENT alert - O ( title?, icon*, text, button) >
<!ATTLIST alert
 id ID #IMPLIED
 type ( warning | caution | note) note
>

<!-- pool
 Elements in the pool are not rendered until they are requested by
identifier reference.  The scope of all resolution of variables, etc.,
is always specified lexically, i.e., variables referenced in the pool
are valid or invalid with respect to the containing scope (mid or
infoContainer), not with respect to the caller's state. Among other
things, the pool may contain elements of the following types: HyTime
location address, HyTime hyperlink, chain, infoContainer, menubar,
pane, alert.  -->

<!ELEMENT pool - O  ANY >

<!-- messageArea
 The contents will be evaluated and concatenated in the order in
which they appear and the results will be rendered by the application
as a "message area" message.  -->

<!ELEMENT messageArea - O ( get | expr | #PCDATA)* >

<!-- clientArea
 A client area is the container for panes, pane groups, and
conditional panes.  The panes are rendered in the order they are
encountered.

NOTE: In an 87268 implementation, the last child of a client area must
be a widget group pane.  This pane is the footer bar.  The members of
the widget group may only be buttons.  The label for the widget group
will not be rendered. -->

<!ELEMENT clientArea - O ( pane | paneGroup | conditionalPane |
alert)* >

<!-- TRANSITIONS & LINKS -->
<!-- gosub, goto, spawn
 Expresses a HyTime hyperlinks with specific MID script traversal
semantics.

Gosub indicates that the state of the current infoContainer be saved
and the target object rendered.  Gosub targets may be of the following
types: infoContainer, chain, pane, conditionalPane, alert, mid.  Gosub
is forbidden to an infoContainer that is nested in another chain.
Gosub is forbidden to a pane or conditionalPane that is nested in
another infoContainer's client area or pane group.

Goto indicates that the current infoContainer be abandoned immediately
and the new infoContainer launched.  Goto targets may be of the
following types: chain, infoContainer, mid.  Goto is forbidden to an
infoContainer nested in another chain.  Return values from objects
which are targets of goto are lost, because there is nothing waiting
on the returned value.  A goto which targets this MID document is
equivalent to a restart of this MID document.

Spawn indicates that control flow splits.  Spawned targets may be of
the following types: infoContainer, chain, pane, conditionalPane,
alert, mid.  Both parent and child compete for focus in the
application display space.  Spawn is forbidden to an infoContainer
nested in another chain.  Spawn is forbidden to a pane nested in a
client area.  Return values from spawned objects are lost, because
there is nothing waiting on the returned value.  When a spawn is
encountered, control stops in the calling script, the target is flowed
until it reaches an idle state, then the caller continues until it
reaches an idle state.  -->

<!ELEMENT ( gosub | goto | spawn) - O ( %locs;)*>
<!ATTLIST ( gosub | goto | spawn)
 HyTime NAME ilink
 HyNames CDATA "linkends target"
 anchrole CDATA "me target"
 target IDREF #REQUIRED
>

<!-- get
 Get expresses that the information at the source be collected,
concatenated, and rendered at the point of the get.

If space is specified, the members of a target aggregate will be
```

```
delimited by a single space before the data is concatenated.  If
normalize is specified, leading and trailing white space is removed,
and multiple contiguous spaces are converted into a single space.  -->

<!ELEMENT get - O ( %locs;)*>
<!ATTLIST get
  HyTime NAME ilink
  anchrole CDATA "me source #AGG"
  HyNames CDATA "linkends source"
  source IDREF #REQUIRED
  space ( space | noSpace) space
  normalize ( normalize | noNormalize) noNormalize
>

<!-- relationship
  The relationship form conforms to the architecture for a HyTime
ilink.  It expresses an authored relationship between two identified
objects.  The application must provide its own element and attribute
declarations for hyperlinking according to the HyTime standard.  This
pseudo-declaration is provided as a model for the HyTime ilink.  The
generic identifier of the relationship governs the relationship
semantic.

The traversal semantic of the relationship is governed by the
traversal attribute.  If traversal is set to be undefined, traversal
decisions will be left up to the application.

Attributes may be added to change traversal from hotspot marking
(interrupt) to hotspot information by request only (polling).  This
would prevent hotspot clutter in an on-line index, for example.

<!ELEMENT relationship - O ( title, %locs;*) >
<!ATTLIST relationship
  HyTime NAME ilink
  id ID #IMPLIED
  relationshipName #CDATA #FIXED
  anchrole CDATA #FIXED "antecedent #AGG consequent #AGG"
  linkends IDREFS #REQUIRED
  extra NAMES #IMPLIED
  intra NAMES #IMPLIED
  endterms IDREFS #IMPLIED
  aggtrav NAMES agg
  MID NAME #FIXED relationship
  privTrav NAMES #IMPLIED
  traversal ( gosub | spawn | goto | undefined) spawn
>
-->

<!-- CONTROLS -->
<!-- menu
  This element declares a named and labeled menu of menus and menu
items.  If disable is specified, the menu label will be visible but
the menu will be inaccessible ("grayed out").  The menu will be
rendered when its label is selected from a rendered parent menu or
menubar.  -->

<!ELEMENT menu - O ( label, ( menu | button | buttonGroup)* ) >
<!ATTLIST menu
  id ID #IMPLIED
  enable ( enable | disable) enable
>

<!-- setState
  This element indicates that the state of the target object should be
modified according to the attributes and content specified.  Possible
targets: menubar, menu, button, buttonGroup.  More complicated
substitutions should use the functionality provided by
conditionalWidget.

The toggle attribute tells whether the target should be toggled on,
toggled off, that the toggle should be removed, or that no change to
the toggle should take place.

The enable attribute tells whether the target should be enabled or
disabled ("grayed out") or that no change should be made.

The action attribute tells whether to modify the target, to remove the
target from its position, or to reset the target to its initial
settings.

The content attribute tells how to treat the content of the setState
element.  The subelements may be inserted before the target, after the
target, or replace the target entirely.  Replacing items on the menubar
with a buttonGroup is not allowed.  -->

<!ELEMENT setState - O ( menu | button | buttonGroup)* >
<!ATTLIST setState
  target IDREFS #REQUIRED
  toggle ( toggleOn | toggleOff | removeToggle | noToggleChange)
noToggleChange
  enable ( enable | disable | noEnableChange) noEnableChange
  action ( modify | remove | reset) modify
  content ( insertBefore | insertAfter | replace) replace
>
```

```
<!-- button
  A button represents a user interface activation control.  The script
is run when the button is activated.  If specified, the name of the
button must be unique within a button group.  If toggleOn is
specified, the button is rendered with a "toggled on" representation.
If toggleOff is specified, the button is rendered with a "toggled off"
representation.  If disable is specified, the button will be visible
but it will be inaccessible ("grayed out").  -->

<!ELEMENT button - O ( label?, script) >
<!ATTLIST button
  id ID #IMPLIED
  name NAME #IMPLIED
  toggle ( toggleOn | toggleOff | noToggle) noToggle
  enable ( enable | disable) enable
>

<!-- fillin
  A fillin represents a fill-in-the-blank widget.  The initial value
of the variable provides the initial text.  When noEcho is
specified, the user's input is not echoed to the display (e.g., for
entering passwords). -->

<!ELEMENT fillin - O ( label, variable) >
<!ATTLIST fillin
  id ID #IMPLIED
  echo ( echo | noEcho) echo
>

<!-- dynamicList
  A dynamicList represents a widget which allows a user to assign a
value to a variable.  When encountered, the label is rendered to name
the widget.  The expr is evaluated; the results become the option
list, and the option list is rendered.  If notRestricted is specified,
the user may enter a value which is not on the option list.  The
script gets run when the user makes a selection. -->

<!ELEMENT dynamicList - O ( variable, label?, expr, script?) >
<!ATTLIST dynamicList
  type ( pickOne | pickMany) pickOne
  restricted ( restricted | notRestricted ) notRestricted
>

<!-- label
  A label is made up of any combination of retrieved text, the results
of evaluation of expressions, parsed character data, and icons.  It is
rendered when its container is rendered, in such a way as to preserve
the semantic of grouping.-->

<!ELEMENT label - O ( get | expr | #PCDATA | icon)* >

<!-- PRIMITIVES & DOCUMENT STRUCTURE -->
<!-- text
  Groups text items.  -->

<!ELEMENT text - O ( get | expr | #PCDATA | specialText | title |
paragraph |
orderedList | unorderedList)* >
<!ATTLIST text
  id ID #IMPLIED
>

<!-- specialTextTypes
  Lists the types of text which are recognized as special. -->

<!ENTITY % specialTextTypes
"visualPunch | foreignWord | semanticStress | newTerm |
bibliographicReference |
wordAsWord | wordAsDefinition | informalName | properObject |
mathExpression |
acronymExpansion | anchor | none" >

<!-- specialText
  Indicates that the contained text is qualified by some semantic.
-->

<!ELEMENT specialText - O ( get | expr | #PCDATA | specialText)* >
<!ATTLIST specialText
  id ID #IMPLIED
  type (%specialTextTypes;) none
>

<!-- title
  Title indicates the title of the object which contains it.  It is
always to be rendered in such a way as to indicate that association.
The contents of the title element are evaluated and concatenated in
the order that they appear.  -->

<!ELEMENT title - O ( get | expr | #PCDATA | specialText)* >

<!-- paragraph
  Indicates the contained text is regarded and rendered as a
paragraph.  -->

<!ELEMENT paragraph - O ( get | expr | #PCDATA | specialText)* >
```

```
<!ATTLIST paragraph
  id ID #IMPLIED
>

<!-- orderedList, unorderedList
  These represent two types of list.  An ordered list is typically
rendered with ascending identifying numbers, letters, etcetera.  An
unordered list is typically rendered with bullets instead.  Items in
either kind of list must be rendered in the order they appear
lexically.  -->

<!ELEMENT ( orderedList | unorderedList) - O ( title?, item+) >
<!ATTLIST ( orderedList | unorderedList)
  id ID #IMPLIED
>

<!-- item
  Represents a list item.  -->

<!ELEMENT item - O ( get | expr | #PCDATA | specialText | orderedList |
unorderedList)* >

<!-- EXTERNAL NOTATIONS -->
<!-- icon, graphic, audio, video, animation
  Access to external notations is made from these elements.  -->

<!ELEMENT ( icon | graphic | audio | video | animation) - O ( get | expr)*
>
<!ATTLIST ( icon | graphic | audio | video | animation)
  id ID #IMPLIED
>

<!-- TABLE ELEMENT TYPES -->
<!-- tableTypes
  The table entity is used to allow the application to substitute any
table type into the pane declaration.-->

<!ENTITY % tableTypes "fcsTable" >

<!-- fcsTable
  The fcsTable conforms to the HyTime finite coordinate space.  It
expresses the abstract layout of a table without imposing assumptions
about how the tabular information will be rendered or used.  To
implement fcs tables, applications must implement the following
architectural forms as required by the HyTime standard: fcs, evsched,
axis, event, extlist, measure, granule.  The element declarations
below are provided as examples that conform to the needed HyTime
architectural forms; they will be recognized by conforming HyTime
engines.

The fcsTable element contains event schedules. Its axisdefs attribute
lists the generic identifiers of the axes that make up the space. The
event elements contained in evscheds are scheduled on each of the
axes
of the fcs.
-->

<!ELEMENT fcsTable - O ( evsched+) >
<!ATTLIST fcsTable
  HyTime NAME fcs
  MID NAME fcsTable
  id ID #IMPLIED
  axisdefs CDATA #FIXED "x y"
>

<!--
Each axis is declared with a specific measurement domain (here,
virspace) and with a specified dimension.

Specific axes with specific axis dimensions must be declared for each
instantiation of table type.  The fcsTable and the evsched refer to
the generic identifier of the desired axis.

NOTE: The axis dimensions "4" and "5" are example dimensions.
Particular tables may have any dimension required, theoretically up to
the the high quantum count value in HyTime.
-->

<!ELEMENT x  - O EMPTY >
<!ATTLIST x
  HyTime NAME axis
  MID NAME axis
  axismeas CDATA #FIXED "virspace"
  axisdim  CDATA #FIXED "4"
>

<!ELEMENT y  - O EMPTY >
<!ATTLIST y
  HyTime NAME axis
  MID NAME axis
  axismeas CDATA #FIXED "virspace"
  axisdim  CDATA #FIXED "5"
>
```

```
<!--
The axisord attribute of the evsched element type dictates the order
in which dimension specifications are to be listed in the extent list
extlist of every event: first the spec for x, then the spec for y.

The #FIXED value of the basegran attribute of evsched establishes a
base measurement unit for scheduling, in this case a "virtual space
unit" ( vsu).
-->

<!ELEMENT evsched - O ( event)* >
<!ATTLIST evsched
  HyTime NAME evsched
  MID NAME evsched
  id ID #IMPLIED
  axisord CDATA #FIXED "x y"
  basegran CDATA #FIXED "vsu"
  gran2hmu NUMBERS #FIXED "1 1"
  overrun ( error | wrap | trunc | ignore) error
>



<!-- event, extlist
  The event and extlist elements are used by fcs but do not require
specific-case architectural form initialization.  They may be used as
they appear below.  -->

<!ELEMENT event - O ( get | expr | #PCDATA)* >
<!ATTLIST event
  HyTime NAME event
  id ID #IMPLIED
  exspec IDREFS #REQUIRED
>

<!ELEMENT extlist - O  ( #PCDATA) >
<!ATTLIST extlist
  HyTime NAME extlist
  id ID #IMPLIED
>

<!-- measure, granule
  The evsched element form requires a measure definition to occur
somewhere in the document.  The following one is an minimal example.

<measure smu=VIRSPACE>
  <granule gn=vsu gd="1 1 VIRSPACE">
</measure>

-->

<!ELEMENT measure - O ( granule+) >
<!ATTLIST measure
  HyTime NAME measure
  id ID #IMPLIED
  smu NAME #REQUIRED
>

<!ELEMENT granule - O EMPTY>
<!ATTLIST granule
  HyTime NAME granule
  gn CDATA #REQUIRED
  gd CDATA #REQUIRED
>

<!-- SEMANTIC GROUPING -->
<!-- paneGroup
  Panes are grouped with some semantic intention of the author.  They
are rendered with a title when encountered.-->

<!ELEMENT paneGroup - O ( title?, ( pane | paneGroup |
conditionalPane)* ) >
<!ATTLIST paneGroup
  id ID #IMPLIED
>

<!-- widgetGroup
  A widgetGroup is an optionally labeled group of widgets.  All of its
contents are rendered when encountered.  The optional script is run
after rendering the widget group, in order that setState may be called
to set the toggled wigets.  -->

<!ELEMENT widgetGroup - O ( label?,
( widgetGroup | conditionalWidget | buttonGroup | dynamicList | button |
fillin)*,
script?) >

<!-- buttonGroup
  Represents a labeled group of buttons.  The semantic of the grouping
is expressed in the type attribute.  An indication of pickOne means
only one button in the group may be selected.  An indication of
pickMany means any number may be selected.  If a default is specified,
the named buttons are preselected.  If no default is specified and the
type is pickOne, the first button is preselected.  -->
```

```
<!ELEMENT buttonGroup - O  ( label?, button*) >
<!ATTLIST buttonGroup
  id ID #IMPLIED
  type ( pickOne | pickMany | button) button
  default NAMES #IMPLIED
>

<!-- menubar
  This element declares a menu bar as a collection of menus and
buttons.  The menubar will be rendered when an infoContainer that
points to it via its menubar attribute is rendered.-->

<!ELEMENT menubar - O ( menu | button)+ >
<!ATTLIST menubar
  id ID #IMPLIED
>

<!-- CONDITIONALS -->
<!-- conditionalPane
  When a conditionalPane is encountered, the expression is evaluated.
Each successive expression is evaluated until one is equal to the
first.  The pane group corresponding to the match is rendered.  If no
expressions match, the final pane group, if present, is rendered as a
default.  This construct is reevaluated when a reflow command is
issued for this element or an element in the proper ancestry of this
element.

NOTE: Conditional panes are intended to be used for alternate displays
of panes, as in alternate language presentation, rather than to
implement 269 preconditions; a conditional pane is not intended to be
the implementation of an if-step, nor is it intended to allow an
entire interactive electronic technical manual to be implemented with
a single infoContainer.  -->

<!ELEMENT conditionalPane - O ( expr, ( expr, paneGroup)*,
paneGroup?) >
<!ATTLIST conditionalPane
  id ID #IMPLIED
>

<!-- conditionalWidget
  When a conditionalWidget is encountered, the first contained
expression is evaluated.  Each successive contained expression is
evaluated, in order, until one is equal to the first.  The widget
group corresponding to the match is rendered.  If no expressions
match, the final contained widget group (if present) is rendered as a
default.  -->

<!ELEMENT conditionalWidget - O ( expr, ( expr, widgetGroup)*,
widgetGroup?) >
<!ATTLIST conditionalWidget
  id ID #IMPLIED
>

<!-- reflow
  When a reflow is encountered, the entire subtree of the target of
the reflow statement is rendered again.  The target must be something
in the current infoContainer, and all current states within the scope
of the infoContainer will be respected.  When no target is specified,
the entire current infoContainer will be rendered again.  If multiple
targets are specified they are rendered again in the order given.  -->

<!ELEMENT reflow - O EMPTY >
<!ATTLIST reflow
  target IDREFS #IMPLIED
>

<!-- SCRIPT ELEMENT TYPES -->
<!-- script
  Scripts are evaluated depending on their context.  First the
declarations are evaluated, then the statements.  The return type for
the script is specified using the functionType attribute.  -->

<!ELEMENT script - O (( vardecl | funcdecl | xenodecl)*, statements) >
<!ATTLIST script
  id ID #IMPLIED
  functionType (%functionTypes;) atom
>

<!-- Declarations
  Declarations are processed and bound to the declared names in the
order the declarations are encountered.  If a variable declaration
initializer contains a reference to another variable, the other
variable must have been declared and initialized prior to the
referring declaration.  -->

<!-- vardecl
  The vardecl element binds a name to a run-time storage location.
Variables must be declared before use.  The expr initializes the
variable.  The variableType attribute specifies the type of the
variable.  Every variable type has a default initialization: zero for
integer and float types, false for boolean, and null for list and
string.  The default sgmlchar is zero.  A local name which is the same
as a name declared higher in the scope stack renders the higher named
```

```
object unreferenceable (the local name is said to "shadow" the higher
one).  -->

<!ELEMENT vardecl - O ( name, expr?) >
<!ATTLIST vardecl
  variableType (%variableTypes;) string
>

<!-- funcdecl
  The funcdecl element binds a function name with argument list, local
state and statement list.  Vardecl names shadow argdecl names.  The
number of arguments, their types, and their order are always fixed.
The functionType attribute specifies the return type of the function.
-->

<!ELEMENT funcdecl - O ( name, argdecl*, vardecl*, statements) >
<!ATTLIST funcdecl
  functionType (%functionTypes;) atom
>

<!-- argdecl
  The argdecl element binds an argument name to a passed value.
Arguments to functions are passed by value.  -->

<!ELEMENT argdecl - O ( name) >
<!ATTLIST argdecl
  variableType (%variableTypes;) string
>

<!-- name
  The name of a function, variable, xenodecl, or scriptLabel is
created by evaluating the contained elements in the order they appear
and concatenating the results.

After the data is concatenated, leading and trailing whitespace
characters are ignored, and multiple whitespace characters are
replaced by a single space.  SGML NAME characters are folded
according
to the NAMECASE GENERAL parameter of the governing SGML
declaration.
-->

<!ELEMENT name O O ( get | expr | #PCDATA)* >

<!-- statements
  Statements are evaluated as directed by the context, in the order
they appear.
NOTE: Although the absence of this container would not create
ambiguities in the MID language (i.e., this container is redundant),
it is provided as a convenience to MID script interpreters.  -->

<!ELEMENT statements O O
( expr | if | loop | break | switch | jump | scriptLabel | goto | spawn
| return | reflow | messageArea | setState)* >

<!-- stringOperations
  The operations specific to string manipulation are collected here.
-->

<!ENTITY % stringOperations "strlen | substr | strcat | fold | isstring"
>

<!-- listOperations
  The operations specific to list manipulation are collected here.
-->

<!ENTITY % listOperations "list | cons | car | cdr | append | isnull | islist |
nth |
count" >

<!-- expr
  The expr element is evaluated as one of its contained elements.  A
copy of the result is returned.

@TBD: The table for interaction between arguments of the various
operators and the return types and exception generation of each has
been left to the implementor of the prototype.  The semantics for implicit
casting have also been left to the implementor of the prototype.
 NOTE: Although the absence of this container would not create
ambiguities in the MID language (i.e., this container is redundant),
it is provided as a convenience to MID script interpreters.  -->

<!ELEMENT expr - O ( assign | variable | constant | function |
add | multiply | subtract | divide | modulus | eq | lt | gt | le | ge | and | or |
ne | not | gettype |
%stringOperations; | %listOperations; | gosub) >

<!-- variable
  The contents of the storage bound to the name are returned to the
caller.  -->

<!ELEMENT variable - O ( name) >

<!-- assign
```

```
  The expression is evaluated and the results are placed in the
variable storage bound to the name.  -->

<!ELEMENT assign - O ( name, expr) >

<!-- function
  The arguments are evaluated in the order in which they appear and
the results are passed as arguments to the named funcdecl or xenodecl.
-->

<!ELEMENT function - O ( name, argument*) >

<!-- argument
  The expression is evaluated as the argument passed to a
function.  -->

<!ELEMENT argument - O ( expr) >

<!-- add, multiply, subtract, divide, modulus, eq, lt, gt, le, ge,
and, or, ne, not
  The expressions are evaluated and the operation is applied according
to the data type.  -->

<!ELEMENT ( add | multiply) - O ( expr+) >
<!ELEMENT ( subtract | divide | modulus) - O ( expr, expr) >
<!ELEMENT ( eq | lt | gt | le | ge | and | or) - O ( expr)+ >
<!ELEMENT ne - O ( expr, expr) >
<!ELEMENT not - O ( expr) >

<!-- constant
  The contents are evaluated and a value of the given constant type is
constructed.  The contantType attribute specifies the data type.

The regular expressions and semantics for the MID primitives are as
follows, with keywords and letters folding case by the rule of the
SGML namecase.

   boolean
     ("true"|"false"|"1"|"0"|"yes"|"no")

"true", 1, and "yes" are equivalent; "false", 0, and "no" are
equivalent.

We declare the following as shorthand:
  DIGIT = ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9")
  NZDIGIT = ("1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9")

   int32, int64
     ("+"|"-")?, NZDIGIT, DIGIT*

These constants are base 10 representation only. "+" indicates
positive; "-" indicates negative.

   uint32, uint64
     NZDIGIT, DIGIT*

 These constants are base 10 representation only.

   float32, float64
     ("+"|"-")?, DIGIT+, ".", DIGIT+ , ( "e", ("+"|"-")?, NZDIGIT, DIGIT* )?

The "e" means "times-ten-to-the".  Digits are required on both sides
of the decimal point.  The mantissa is represented in 16 bits for
float32, and 32 bits for float64.  The remaining bits are reserved for
the exponent.

   sgmlchar
     And any single valid sgml character

   string
     any valid string of sgml characters

NOTE: In string and sgmlchar, record start (RS) and record end (RE)
are ignored according to the rules in ISO 8879.

-->

<!ELEMENT constant - O ( get | #PCDATA)* >
<!ATTLIST constant
constantType (%constantTypes;)   #REQUIRED
normalize ( normalize | noNormalize ) noNormalize
recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter
>

<!-- if, else
  If the expression evaluates to true, the statements in the
statements element are executed.  Otherwise, the statements within the
else are evaluated.  -->

<!ELEMENT if - O ( expr, statements, else?) >
<!ELEMENT else - O ( statements) >

<!-- loop
  The expression is evaluated.  If the expression is true, the
```

```
statements are executed.  The expression is reevaluated and the
statements are reexecuted until the expression returns false.  -->

<!ELEMENT loop - O ( expr, statements) >

<!-- continue, break
  Continue causes execution to resume at the top of the nearest
enclosing loop, whereupon the loop's expr is reevaluated.  As always,
if the expr evaluates to false, execution resumes at the statement
following the loop.  Break causes execution to resume at the statement
following the nearest enclosing loop or switch.  If there is no
lexically enclosing loop, continue is ignored.  If there is no
lexically enclosing loop or switch, break is ignored.  No stacks are
affected.  Jumping to a point within a loop initiates looping
behavior.  Jumping to a point within a switch causes the switch's expr
to be evaluated automatically prior to execution of any statements.
-->

<!ELEMENT ( continue | break) - O EMPTY >

<!-- switch, case, default
  The expression is evaluated.  Each of the expressions of the cases
is evaluated in the order in which the cases appear until one matches
the switch expression.  If a match is found, the statements under the
matched case are executed until a break is encountered.  Control
continues to the statements under the next case if no break is
encountered; the interceding expr is not evaluated.  If no case
expression is matched, the default statements are executed.-->

<!ELEMENT switch - O ( expr, case*, default?) >
<!ELEMENT case - O ( expr, statements?) >
<!ELEMENT default - O ( statements?) >

<!-- jump, scriptLabel
  Control immediately jumps to the named script label.  Certain
restrictions apply: script labels are scoped local to a given
script.  -->

<!ELEMENT jump - O ( name) >
<!ELEMENT scriptLabel - O ( name) >

<!-- gettype
  Returns the type of the expression.  Return type: string.  -->

<!ELEMENT gettype - O ( expr) >

<!-- STRING OPERATIONS -->

<!-- strlen
  Returns the length of the string.  Return type: uint32.  Returns
zero if expression is not a string.  -->

<!ELEMENT strlen - O ( expr) >

<!-- substr
  First expr: string.  Second expr: start position.  Third expr: run
length.  This construct returns the substring of the string, given
start position and run length.  Start position is counted from 1.
Unspecified run length or run length greater than the length of the
string indicates the rest of string.  Returns null string if start
position exceeds string length.  Return type: string.  -->

<!ELEMENT substr - O ( expr, expr, expr?) >

<!-- strcat
  Returns the concatenation of the strings.  Expressions in this
element's immediate content which evaluate to non-strings are ignored.

If space is specified, a space character is inserted between
expressions.  If normalize is specified, leading and trailing white
space is removed, and multiple contiguous spaces are converted into a
single space.

Return type: string.  -->

<!ELEMENT strcat - O ( expr)+ >
<!ATTLIST strcat
  space ( space | noSpace ) noSpace
  normalize ( normalize | noNormalize ) noNormalize
  recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter

>

<!-- fold
  Returns the folded version of the string.  Converts string to
uppercase using SGML name case folding rules.  The name attribute
tells whether the name characters are to be folded according the SGML
declaration rules for entity names or for general names.  Return type:
string.  -->

<!ELEMENT fold - O ( expr) >
<!ATTLIST fold
  name ( general | entity) general
  normalize ( normalize | noNormalize ) noNormalize
  recordDelimiter (recordDelimiter | norecordDelimiter) recordDelimiter
```

```
>

<!-- isnull
  This function returns true if the expression is a null (empty)
string or a null list.  It returns false otherwise.  Return type:
boolean.  -->

<!ELEMENT isnull - O ( expr) >

<!-- isstring
  Returns whether the expression is a string.  Return type: boolean.
-->

<!ELEMENT isstring - O ( expr) >

<!-- LIST OPERATIONS -->
<!-- list
  Each expression in the content of this element is evaluated and
becomes an top-level item on the returned list.  If no expressions are
specified, this element returns a null list. A list in MID has the
same binary tree implementation as lists in Lisp or Prolog.  Return
type: list.  -->

<!ELEMENT list - O ( expr)* >

<!-- cons
  This function returns a list in which the result of evaluating the
first expression is prepended to the list found in the second
expression.  The second expression must be a list.  Return type: list.

NOTE: The names cons, car, and cdr, while perhaps non-intuitive in
English, is precisely meaningful in LISP or Prolog; they were chosen
deliberately to enhance interdisciplinary communications.  -->

<!ELEMENT cons - O ( expr, expr) >

<!-- car
  This function returns the car of the list, i.e, the first item of a
cons pair.  The expression must be a list.  Return type: any.  -->

<!ELEMENT car - O ( expr) >

<!-- cdr
  This function returns all but the first item in a list (the second
half of a cons pair).  Return type: list.  -->

<!ELEMENT cdr - O ( expr) >

<!-- append
  This function returns the results of appending a list to a list.
Return type: list.  -->

<!ELEMENT append - O ( expr, expr) >

<!-- islist
  This function returns true if the expression is of type list.
Return type: boolean.  -->

<!ELEMENT islist - O ( expr) >

<!-- nth
  expr1: list.  expr2: integer.  Returns the nth item of the list,
counting from one, without recurring into nested lists.  Returns the
null list if there is no such item.  Return type: atom or list.  -->

<!ELEMENT nth - O ( expr, expr) >

<!-- count
  Returns the number of items in the given list, without recurring
into nested lists.  Returns zero if expression is a list which has no
members.  Return type: uint32. -->

<!ELEMENT count - O ( expr) >

<!-- EXTERNAL PROCESS -->
<!-- xenodecl
  The xenodecl element binds a name and argument declarations to a
call to an external notation.  The functionType attribute specifies
the return type of this construct.-->

<!ELEMENT xenodecl - O ( name, argdecl*, xeno) >
<!ATTLIST xenodecl
  functionType (%functionTypes;) atom
>

<!-- xeno
  The xeno element represents a subclass of the HyTime notloc.
Argument names from the containing xenodecl indicate substitution into
the RCDATA of the xeno when the argument name is surrounded by the
string tokens specified in argBegin and argEnd.-->

<!ELEMENT xeno - O RCDATA >
<!ATTLIST xeno
```

```
  HyTime NAME notloc
  id ID #IMPLIED
  qdomain IDREFS #IMPLIED
  qcontext IDREF #IMPLIED
  ordering ( ordered | noorder) noorder
  set ( set | notset) notset
  aggloc ( aggloc | agglink | nagg) nagg
  argBegin CDATA "$("
  argEnd CDATA ")"
>

<!-- return
  This element terminates processing of the nearest containing
construct specified by the construct attribute, and returns the value
resulting from evaluating the expression.  If there is no expression,
the return value is the default initialization for the stated return
type.  -->

<!ELEMENT return - O ( expr?) >
<!ATTLIST return
  construct ( mid | chain | infoContainer | pane | alert | script | function)
function
>

<!-- HYTIME -->
<!-- security
  Security is an implementation of the HyTime activity form.  The
security attribute tells what level of security.  Elements mid and
pane may point to a security element, thereby indicating the security
level.  The contained script (if any) will be run when the indicated
activity (in this case, access) occurs.-->

<!ELEMENT security - O (script)? >
<!ATTLIST security
  id ID #IMPLIED
  HyTime NAME activity
  actypes NAMES access
  level (unclassified | confidential | secret | topSecret) unclassified
>

<!-- The following HyTime location types are instantiated directly
from the HyTime standard.  -->
<!ELEMENT nameloc - O ( nmlist | HyQ)* >
<!ATTLIST nameloc
  HyTime NAME nameloc
  id ID #REQUIRED
  ordering ( ordered | noorder) noorder
  set ( set | notset) notset
  aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT nmlist - O ( #PCDATA) >
<!ATTLIST nmlist
  HyTime NAME nmlist
  nametype ( entity | element | unified) #REQUIRED
  obnames ( obnames | nobnames) #REQUIRED
  docorsub ENTITY #IMPLIED
  dtdorlpd NAMES #IMPLIED
>

<!ELEMENT HyQ - O ( #PCDATA) >
<!ATTLIST HyQ
  HyTime NAME nmquery
  qdomain IDREFS #IMPLIED
  qcontext IDREF #IMPLIED
  notation NAME #FIXED HyQ
  delims CDATA #IMPLIED
  fn NAME #IMPLIED
  usefn NAME #CONREF
  args IDREFS #IMPLIED
  qpnpsn NAMES #IMPLIED
  qltnlmgi NAMES #IMPLIED
>

<!ELEMENT treeloc - O ( marklist*) >
<!ATTLIST treeloc
  HyTime NAME treeloc
  id ID #REQUIRED
  overrun ( error | wrap | trunc | ignore) error
  treecom ( treecom | ntreecom) ntreecom
  locsrc IDREFS #IMPLIED
  ordering ( ordered | noorder) noorder
  set ( set | notset) notset
  aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT relloc - O ( dimlist*) >
<!ATTLIST relloc
  HyTime NAME relloc
  id ID #REQUIRED
  root IDREFS #IMPLIED
  relation ( anc | esib | ysib | des | parent | children) parent
  overrun ( error | wrap | trunc | ignore) error
  locsrc IDREFS #IMPLIED
```

```
 ordering ( ordered | noorder) noorder
 set ( set | notset) notset
 aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT dataloc - O ( dimlist*) >
<!ATTLIST dataloc
 HyTime NAME dataloc
 id ID #REQUIRED
 quantum ( str | norm | word | name | sint | date | time | utc) str
 catsrc ( catsrc | nocatsrc) nocatsrc
 catres ( catres | nocatres) nocatres
 overrun ( error | wrap | trunc | ignore) error
 locsrc IDREFS #IMPLIED
 ordering ( ordered | noorder) noorder
 set ( set | notset) notset
 aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT marklist O O ( marklist | #PCDATA)* >
<!ATTLIST marklist
 HyTime NAME marklist
>

<!ELEMENT dimlist O O ( dimlist | marklist | #PCDATA)* >
<!ATTLIST dimlist
 HyTime NAME dimlist
>

<!ELEMENT proploc - O ( qpn | #PCDATA) >
<!ATTLIST proploc
 HyTime NAME proploc
 id ID #REQUIRED
 joint ( joint | several) several
 apropsrc ( apropsrc | solesrc) solesrc
 notprop ( error | empty | ignore) ignore
 locsrc IDREFS #IMPLIED
 ordering ( ordered | noorder) noorder
 set ( set | notset) notset
 aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT qpn - O ( pn, spec?)+ >
<!ATTLIST qpn
 HyTime NAME qpn
 id ID #REQUIRED
>

<!ELEMENT pn - O RCDATA >
<!ATTLIST pn
 HyTime NAME pn
>

<!ELEMENT spec - O (( qpn | qltn)+ | pval) >
<!ATTLIST spec
 HyTime NAME spec
>

<!ELEMENT qltn - O RCDATA>
<!ATTLIST qltn
 HyTime NAME qltn
>

<!ELEMENT pval - O RCDATA >
<!ATTLIST pval
 HyTime NAME pval
>

<!ELEMENT notloc - O ANY >
<!ATTLIST notloc
 HyTime NAME notloc
 id ID #REQUIRED
 qdomain IDREFS #IMPLIED
 qcontext IDREF #IMPLIED
 fn NAME #IMPLIED
 usefn NAME #CONREF
 args CDATA #IMPLIED
 ordering ( ordered | noorder) noorder
 set ( set | notset) notset
 aggloc ( aggloc | agglink | nagg) nagg
>

<!ELEMENT bibloc - O ANY >
<!ATTLIST bibloc
 HyTime NAME bibloc
 id ID #REQUIRED
 qdomain IDREFS #IMPLIED
 qcontext IDREF #IMPLIED
 fn NAME #IMPLIED
 usefn NAME #CONREF
 args CDATA #IMPLIED
>
```

B. Relationship example (for illustration only)

First consider the modification of < **relationship**> to look like:

```
<!ELEMENT relationship - O ( title,(%locs;)*)>
<!ATTLIST relationship
  HyTime NAME ilink
  MID NAME #FIXED relationship
  relationshipName #CDATA #FIXED
  id ID #IMPLIED
  anchrole CDATA #FIXED "antecedent #AGG consequent #AGG"
  linkends IDREFS #REQUIRED
  privTrav NAMES #IMPLIED
  extra NAMES #IMPLIED
  intra NAMES #IMPLIED
  endterms IDREFS #IMPLIED
  aggtrav NAMES agg
  traversal ( gosub | spawn | goto | undefined) spawn
>
```

The *relationshipName* is a displayable string that indicates the purpose of each element that is derived from the relationship form.

The *privTrav* contains none, one, or all of the anchrole names. Its purpose is to define a default traversal from none, all, or each linkend to another linkend within the relationship.

Now here's an example.We create a relationship element called 'equipment' that will represent any single equipment component of a subsystem. The 'subsystem' containing the equipment will be a second relationship. Each equipment - in this case a radio and a fire extinguisher - has a common set of contexts (that the author has defined) where references to equipment are found. In addition, groups of equipment may be found in subsystems.

For example, the author plans to identify the 'AN/ABC Radio Set' equipment in the context of descriptive text, photos, schematic diagrams, and a parts list. Similarly, he will identify 'NoFire Model 42' fire extinguisher as an equipment element in text, photo, schematic, and parts list. For each equipment, there will also be a list of hotspots (in text and graphics) that refer to the same equipment. There can be multiple schematics and hotspots, as indicated by the #AGG (which specifies that the link is allowed to be aggregate, e.g., a **nameloc** with a namelist containing multiple IDREFs).

Here's what the DTD looks like for the equipment **relationship** element:

```
<!ELEMENT equipment - O (title,(%locs;)*)>
<!ATTLIST equipment
  HyTime NAME ilink
  MID NAME #FIXED relationship
  relationshipName #CDATA #FIXED "Component"
  id ID #IMPLIED
  anchrole CDATA #FIXED "descriptiveInfo
                         photo
                         schematics #AGG
                         partList
                         hotspots #AGG"
```

```
  linkends IDREFS #REQUIRED
  privTrav NAMES #IMPLIED
  extra NAMES #ALL
  intra NAMES #ALL
  endterms IDREFS #IMPLIED
  aggtrav NAMES agg
  traversal ( gosub | spawn | goto | undefined) spawn
>
```

There are hotspots, for both the fire extinguisher and the radio, in the text of an **alert**. However, the hotspot for the radio actually refers to the power switch of the radio. Therefore, we wish the traversal from the power switch hotspot to take us to a particular location on the radio photograph, not just the graphic pane containing the photo. This requires us to got through a notation to bring the photo objects from Microsoft SHG format into the MID name space. To make this less obtuse, we create a second relationship type called directRel that allows us to specify a preferred traversal from the hotspot to the powerswitch object on the photo. Note that the equipment relationship can only specify a privTrav from the set of hotspots to the photo, not from a specific hotspot to an object on the photo. Applications must sort out the possible ambiguity of multiple privTravs on the same anchor, but from different relationships, having conflicting traversal priorities.

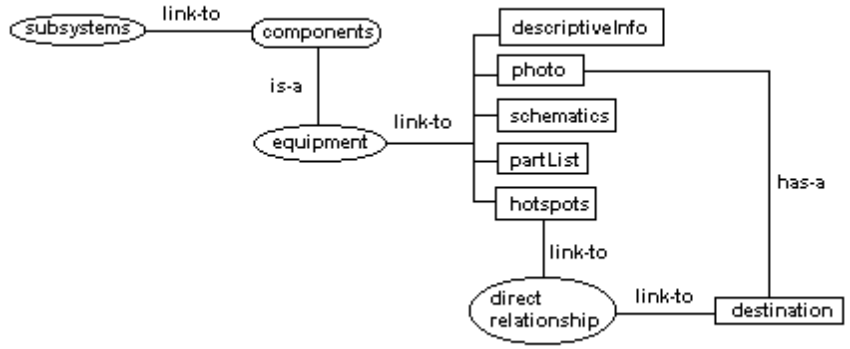
```
<!ELEMENT directRel - O (title,(%locs;)*)>
<!ATTLIST directRel
  HyTime NAME ilink
  MID NAME #FIXED relationship
  relationshipName #CDATA #FIXED "Direct Link"
  id ID #IMPLIED
  anchrole CDATA #FIXED "source destination”
  linkends IDREFS #IMPLIED
  privTrav NAMES #IMPLIED
  extra NAMES “A E”
  intra NAMES “A E”
  endterms IDREFS #IMPLIED
  traversal (gosub | spawn | goto | undefined) spawn
>
```

Now, moving up a level to the subsystem definition, the author creates a third relationship element that specifies that a particular set of equipment (subsystem components) belongs to a subsystem. We will define a specific subsystem for Emergency Management that will contain both the radio and the fire extinguisher. Here is the DTD element:

```
<!ELEMENT subsystem - O (title?, %locs;*)>
<!ATTLIST subsystem
  HyTime NAME ilink
  MID NAME #FIXED relationship
  relationshipName #CDATA #FIXED "Subsystem"
  id ID #IMPLIED
  anchrole CDATA #FIXED " components #AGG"
  linkends IDREFS #IMPLIED
  privTrav NAMES #IMPLIED
  extra NAMES #ALL
  intra NAMES #ALL
  endterms IDREFS #IMPLIED
  aggtrav NAMES agg
  traversal (gosub | spawn | goto | undefined) spawn
>
```

The ‘schematics’ and ‘hotspots’ linkends in the equipment element, and the ‘components’ linkend in the subsystem element, actually comprise a **nmlist** of equipment elements.

Here is what we have so far:



Hence, the instance looks something like this:

```
...
<pool>

<!-- These are the relationships -->

<equipment id=anabc linkends="abcText abcPhoto abcWiringDiagrams abcPartList
abcHotspots">
  <title>AN/ABC Radio Set</title>
  <!-- list of schematics #AGG -->
  <nameloc id=abcWiringDiagrams>
    <nmlist nametype=element obnames=nobnames >
        abcWiring01of03 abcWiring02of03 abcWiring03of03 </nmlist>
  </nameloc>
  <!-- list of hotspots #AGG -->
  <nameloc id=abcHotspots>
    <nmlist nametype=element obnames=nobnames >
        abc001 abc002 abc003 abc004 </nmlist>
  </nameloc>
</equipment>

<equipment id=nofire linkends="nofireText nofirePhoto nofireSchems nofirePartList
nofireHotspots">
  <title>NoFire Model 42</title>
  <nameloc id=nofireSchems>
    <nmlist nametype=element obnames=nobnames >
        nofireSchem1 nofireSchem2 </nmlist>
  </nameloc>
  <nameloc id=nofireHotspots>
    <nmlist nametype=element obnames=nobnames >
        nofire001 nofire002 nofire003 nofire004 </nmlist>
  </nameloc>
</equipment>

<!-- ‘naked’ pane in the pool -->
<pane id=nofireTextPane>
  <text id=nofireText>
  The NoFire Model 42 Fire Extinguisher puts out fires and spews chemicals like a
champ.
  </text>
```

```
</pane>

<subsystem id=emSubsystem linkends="emNames" traversal=gosub>
  <title>Emergency Management</title>
  <nameloc id=emNames>
    <nmlist nametype=element obnames=nobnames >
        anabc nofire </nmlist>
  </nameloc>
</subsystem>


<!-- Here is the part that does the directRel from the specific hotword in the
alert to the power switch object in the photo, called 'powerswitch' -->

<!-- SHG is Microsoft file format that embeds named coordinate zones in a Windows
bitmap .BMP -->
<!NOTATION SHG PUBLIC
   "-//ISBN 0-7923-9432-1::Graphic Notation//NOTATION
    Microsoft Segmented Hypermedia//EN">

<!NOTATION SHGNAMES PUBLIC
   "-//MIDcommittee//NOTATION
    shgnames//EN">

<!ENTITY abcShgFile SYSTEM "radioabc.shg" NDATA SHG>

<!-- Give the entity an ID -->
<nameloc id=abcGraphic>
  <nmlist nametype=entity obnames=nobnames>
    abcShgFile</nmlist></nameloc>

<!-- This pane is an anchor for equipment 'anabc' -->
<pane id=abcPhoto>
  <graphic id=renderGraphic>
    <get target=abcGraphic></graphic></pane>

<notloc id=abcPowerSwitchLoc notation="SHGNAMES" qdomain=abcGraphic>
    powerswitch</notloc>

<!-- Here is the relationship directRel defining immediate traversal to the power
switch location graphic, even though the linkend abc001 is also listed as a
linkend of the equipment relationship -->

<directRel
    id=abc001powerswitch
    linkends="abc001 abcPowerSwitchLoc"
    privTrav=destination
    endterms="#NONE abcPowerSwitchLoc">
  <title>AN/ABC Radio Power Switch Location</title>
</directRel>

</pool>

...
<!-- We somehow get to a repair procedure -->

<chain id=engineRepair>
...

<infoContainer id=engineRepairStep006>
```

```
  <title>Removing the Battery</title>
  <clientArea>
    <alert id=a004 type=warning>
      <title>Spark Hazard
      <text>When removing the battery terminal connectors,
            there is a chance that a spark will be generated.
            Be sure you are familiar with the location of a trusty
            <specialText id=nofire001 type=anchor>
            NoFire Model 42 Fire Extinguisher</specialText>
            and the power switch of the shipboard
            <specialText id=abc001 type=anchor>AN/ABC Radio Set</specialText>.
      </text>
    </alert>
  <pane>
...
</infoContainer>

...
```

The **specialText** element `nofire001` is an anchor for one of the `nofireHotspots` links in the `equipment` **relationship** `nofire`.

The **specialText** element `abc001` is both (1) an anchor for one of the `abcHotspots` links in the `equipment` **relationship** `anabc`, and (2) an anchor for the `source` linkend of the `directRel` **relationship** `abc001powerswitch`. Because the `directRel` has a `privTrav` that applies (based on the `endterms`) to traversal from the hotword to the graphic coordinate zone, this traversal will take precedence over those defined by the equipment relationship.

What the Browser (MIDReader) Application does with all this stuff.

A browser application could set the primary method for activating a link as a left mouse button click. Additional information can be gathered by clicking the right mouse button.

For the example above, a user might get the same action from left or right mouse-clicks on the `nofire001` hotword - a list of possible traversals from the information in the table below.

relationshipName from DTD	contents of the relationship title	anchrole
Component	NoFire Model 42	descriptiveInfo
Component	NoFire Model 42	photo
Component	NoFire Model 42	schematics
Component	NoFire Model 42	partList
Component	NoFire Model 42	hotspots
Subsystem	Emergency Management	components

In the case of the `abc001` hotword, the left mouse-click might launch a graphic pane containing the AN/ABC Radio `photo`, with the `powerswitch` object highlighted. A right-click would produce a list of possible traversals similar to the one above:

relationshipName from DTD	contents of the instance of relationship title	anchrole

Component	AN/ABC Radio Set	descriptiveInfo
Component	AN/ABC Radio Set	photo
Component	AN/ABC Radio Set	schematics
Component	AN/ABC Radio Set	partList
Component	AN/ABC Radio Set	hotspots
Subsystem	Emergency Management	components
Direct Link	AN/ABC Radio Power Switch Location	destination

At this point, it is left to the application to make a seamless connection between the abcPowerSwitchLoc (or the powerswitch object in the graphic through some other method than **notloc**), and the graphic window that will display it (i.e., the application should launch its own window given a named element in a known graphic entity).

C. MID Background

C.1 Background

The purpose of a metafile for interactive documents is to embed behavior in an Interactive Electronic Technical Manual (IETM) document, and to improve portability and reuse of that document. The metafile enables IETM documents to be transferred from dissimilar authoring systems for unambiguous presentation and interaction on dissimilar display systems.

C.1.1 Interactive Electronic Technical Manual

An IETM, as defined in the DoD IETM Specifications, is a package of information required for the diagnosis and maintenance of a weapons system, optimally arranged and formatted for interactive screen presentation to the end-user. It is a Technical Manual prepared (authored) by a contractor and delivered to the Government, or prepared by a Government activity, in digital form on a suitable medium, by means of an automated authoring system. An IETM is designed for electronic screen display to an end user, and has the following three characteristics:

1. The information is designed and formatted for screen presentation to enhance comprehension.
2. The elements of technical data making up the TM are interrelated. A user's access to required information is possible by a variety of paths.
3. The computer-controlled TM display device functions interactively (as a result of user requests and information input) to provide procedural guidance, navigational directions, and supplemental information.

IETMs allow a user to locate required information faster and more easily than is possible with a paper technical manual. They are easier to comprehend, more specifically matched to the system configuration under diagnosis, and are available in a form that requires much less physical storage than paper. Powerful interactive troubleshooting procedures, not possible with paper technical manuals, can be made available using the intelligent features of the IETM display device.

C.1.2 Metafile for Interactive Documents

MIL-M-87268 and MIL-D-87269 define the process for authoring and displaying IETMs. They implement an underlying strategy that separates the IETM source data base from the electronic display of the formatted IETM. The roles of these specifications are as follows:

- MIL-M-87268 defines how the IETM should look and behave to the reader.
- MIL-D-87269 establishes the IETM database forms, structure, and key controlling mechanisms.

This process has found favor in the IETM development community. Most DoD IETM applications separate the presentation attributes from the IETM content. However, since the standardization focus has been on the database data structures and not on the run-time version of the IETM, (the View Package), each software vendor has developed a proprietary format for capturing and moving the IETM to the Presentation System. In attempting to maintain a flexible approach to the definition of the IETM data base, the specifications nearly guarantee non-portability across different vendor products.

The current specifications define the data format and contents for the IETM database, and define the targeted "look and feel" for the Presentation System. What has not been defined is how the ported IETM looks, i.e., what the Presentation System reads as the IETM.

To completely specify the electronic pathway for the process of preparing and using IETMs, there must also be an unambiguous definition of the "portable IETM" -- the data that is produced by an Authoring System from the source data and read by the Presentation System for display. IETM Presentation Systems must be able to take this "neutral" data from varying authoring systems and structure it for display on dissimilar Presentation Systems with a minimum amount of human intervention. This is the place for the MID.

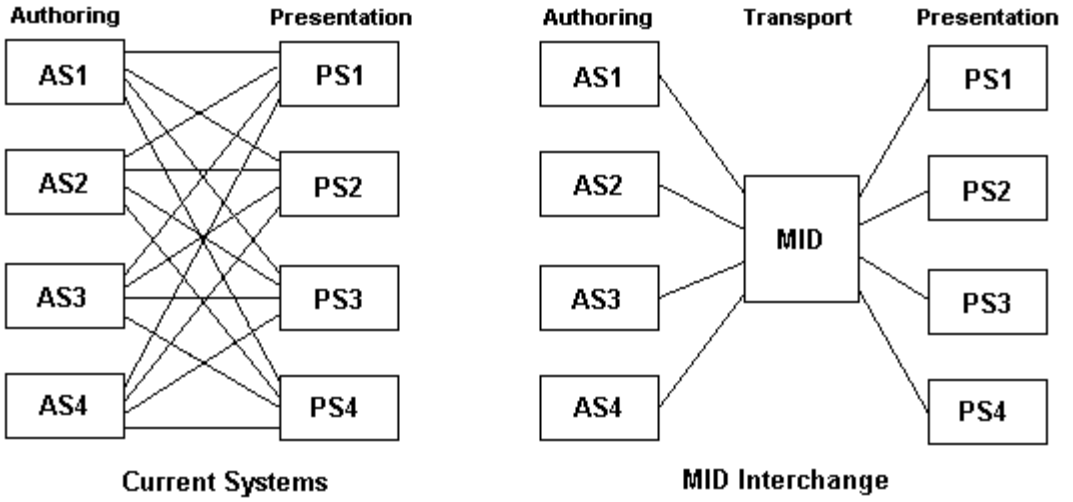


Figure C-1. Current and Future IETM Systems

As shown in Figure C-1, an Authoring System currently must generate a separate version of an IETM for each targeted Presentation System. It is obvious that a common interchange structure simplifies both the Authoring Systems and the Presentation Systems.

The structure for the ported IETM is called the Metafile for Interactive Documents (MID). The MID provides target structures for the authoring systems to write to, and for the Presentation Systems to read from. It is an intermediate structure that, once specified, completes the IETM process, as shown in the second part of Figure C-1 and in Figure C-2.

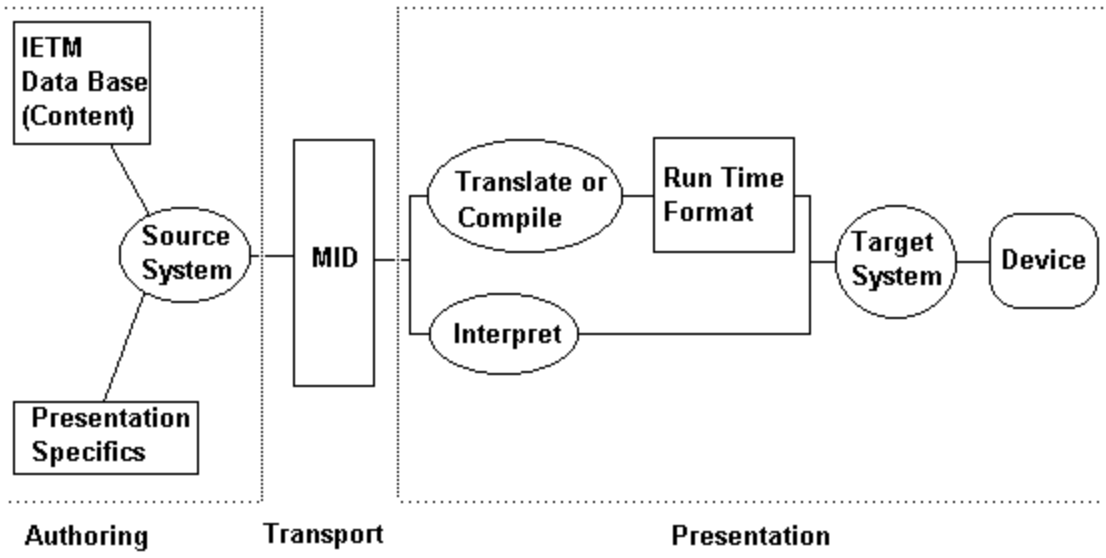


Figure C-2. The IETM Process

The Source System exports a MID instance which is transferred to the Translation, Compilation or Interpretation software in the Presentation system. The output from the Compilation or Translation software may be encoded in a Run-Time structure that is read by the Target System and presented on the Device. Otherwise, a Target system may interpret and present the MID instance directly.

C.1.3 Goals and Objectives

The objective of this effort is to develop for the U.S. Navy, a rigid and precise data format which contains or references all the content from an IETM data base developed in accordance with MIL-D-87269 or other specified definitions, but structured to contain all the sequencing information necessary to unambiguously specify the behavior of that information when presented. Use of the MID will ensure interoperability between two or more independently developed MID instances which refer to information in the other using an external reference format specifed in this document and based on the international standard for location and addressing in hypermedia, ISO 10744 - HyTime. This will result in an unambiguous, complete, precise and consistent presentation of the IETM information across dissimilar authoring tools and Presentation Systems.

C.2 MID REQUIREMENTS

The first step in the MID development was to establish the following requirements for the MID specification:

1. A MID can provide a portable container of data from the document database (e.g., MIL-D-87269) suitable for the Presentation System (e.g., MIL-M-87268).
2. A MID must enable an implementor to create a very simple, yet efficient presentation of IETM information on any delivery device that is capable of the most common graphical user interface interaction and display operations.
3. A MID can be disconnected from the document database without losing essential content.
4. A MID must be readable and displayable in many common hardware/software environments.
5. A MID does not need to be editable in its defined format.
6. A MID must be based on accepted international standards (e.g., SGML, HyTime).
7. A MID must be implementable.
8. A MID must be based on current technology.
9. A MID is targeted to multimedia presentation of complex information.
10. A MID is targeted to a single and deliberately simple presentation. Different MID instances may be developed for different MIL-D-87269 content layers (each utilizing standard MID element types) or for information that is constrained by non-MIL-D-87269 databases and data in non-SGML notations.
11. A MID separates those attributes of GUI design that are typically specific and proprietary to the vendors from those that are similar across GUIs.
12. A MID must be extensible (e.g., include a process (an "escape" mechanism) to accommodate new component types).
13. A MID is independent of specific authoring and Presentation Systems.
14. A MID is unambiguous; it must provide sufficiently explicit definition of data types and execution semantics so as to enable unambiguous IETM presentations on differing target display systems.
15. While source files must be transportable to MID files, there is no requirement that MID files must be transportable back to the original source.
16. The potential functionality of the MID must encompass all functionality defined in MIL-D-87269 and MIL-M-87268.

C.3 THE MID ARCHITECTURE

The role of the MID is to provide a language for authoring and transporting "intelligent" documents. The MID architecture enables an author to determine where and how much that "intelligence" is used to direct a user in locating and interacting with the information. While the presentation format and behaviors of the "intelligent" document are standardized by a MID, the MID is free of complex content structures. All information presented in a MID is represented by a small set of content primitives. This enables the MID to be used for many hypermedia applications.

C.3.1 Overview

The MID provides a modular approach to authoring and maintaining IETMs. A MID standardizes the presentation of information and the behavior of that presentation across platforms. This is achieved through a standard set of user interface objects combined with an internal scripting language that controls the interaction of these objects with each other and the user as the objects access databases and display information on a Presentation System.

Cross-platform interoperability is achieved through the use of SGML/HyTime. The MID is an application of SGML (ISO 8879) and HyTime (ISO 10744). SGML standardizes the syntax of the Document Type Definition for the MID language. HyTime provides standard models for location and addressing element types used in the MID DTD. This document assumes that the reader is familiar with the concepts and requirements of SGML.

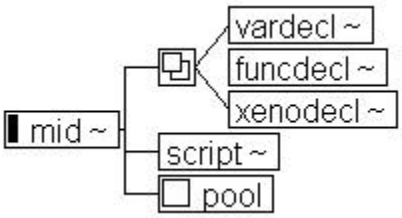


Figure C-3. MID Document Element Type

C.3.1.1 Using MID Scripts

A Presentation System starts a MID interactive session based on the contents of the MID Master Script in the document instance. As the Master Script assumes control of the session, any < **infoContainer**> automatically returns to the Master Script if no other link is provided in the < **infoContainer**> and executes the next statement in the Master Script. Figure C-4 illustrates the components of a < **script**> element type.

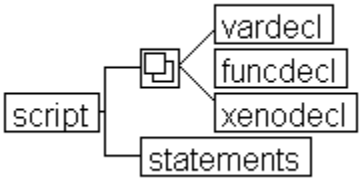


Figure C-4. MID Script Element Type

The example script below contains a < **gosub**> link which is processed. The result of the first < **gosub**> sets a "choice" **variable** that was declared as an application global **variable**. Next, a **switch** statement is executed that contains two <**case**> statements which are evaluated to determine which of two < **goto**> links are traversed. Upon returning to the

Master Script, another < **gosub**> is executed. All other processing is determined by the instructions encapsulated in each <**infoContainer**>.

```
<script><statements>
  <gosub target=i1>
     <switch><expression><variable><name>choice
        <case><expression><constant>House
     <statements><goto target=i2></statements>
        </case>
        <case><expression><constant>Automobile
     <statements><goto target=i3></statements>
       </case>
     </switch>
  <gosub target=i4>
</script>
```

The complexity of a MID Master Script depends on the overall complexity of the set of MID document entities, the relationships among them, and the mission of the IETM. MID allows for flexible approaches to encoding an IETM as many different sources can be used which have different requirements for the levels of directing the navigation of their content. For example, a training script may constrain the user of the IETM to view information in a specific order; whereas, a technical manual may allow browsing in parts of the IETM at will.

Another example involves a document containing several traditional volumes of information. A MID can control access to multiple subsets of the volumes by a master index which enables a user to determine which volume has the information required. The initial Master MID could be a very simple script that functions as an Index of Volumes with hyperlinked content tables; or, it could be a very complex script that uses interactive dialogs to determine what information is needed, locates that information and presents it. How the author distributes the "intelligent" aspects of a document is a matter of style, efficiency of processing and the phase of document creation.

C.3.1.2 Using Application Global Declarations

Declarations in MID are scoped by the major element within which they are declared. Application globals, however, declare the global functions and variables that are available to any function that requires them during execution of a MID script. Figure C-5 shows the types of application globals:

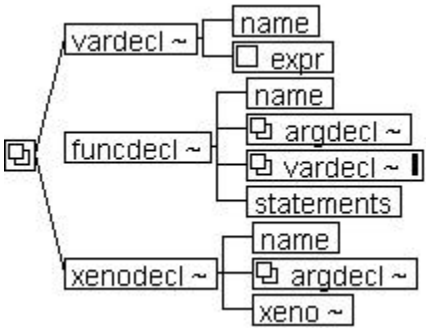


Figure C-5. Applications Globals

The **variable** declaration, < **vardecl**> declares a <type> for the **variable**. The <type> is a string. Elements for MID function declarations (**funcdecl**), and declarations for non-MID functions (**xenodecl**) are also included in the application globals. ***Note: The current revisions to the MID has changed and eliminated the use of "application globals" as mentioned in this section.***

C.3.1.3 Using Information Containers

For the MID user, the Information Container is the locus of interaction. An Information Container has user interface objects and content that are presented to the user and managed by the processing of **infoContainer**s.

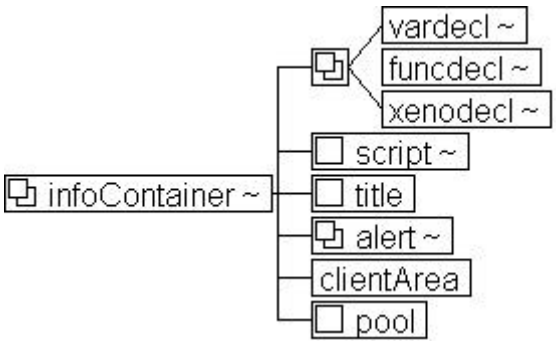


Figure C-6. Information Container Element Type

Although it is convenient to divide the information by screens, the MID author should note that one < **infoContainer**> does not always equate to one screen. For example, an **alert** may be displayed prior to imaging the panes in a client area. All of the processing is in one < **infoContainer**>, although to the human observer, they might appear to be separate screens. For this reason it is more useful to think of an < **infoContainer**> as encapsulating a state space or unit of process although it is more natural to consider it a unit of presentation.

C.3.1.4 Using Pools

Pools enable the author to reuse common dialogs and **alert**s. From within a script, the author can use an element in the **pool** by linking to it using a < **get**>, then return to the place in the script from which the **pool** element was called.

A common application of a < **pool**> is to hold all of the Warnings, Cautions and Notes that are reused extensively throughout an IETM.

C.3.1.5 Using Queries to Access Document Databases

In the MID DTD, the following construct is often found in the content model of element types.

get | #PCDATA | script

The capability to determine the target of a link based on the evaluation of a script is referred to as "dynamic hyperlinking." The SGML OR group shown above provides the basic structure which defines dynamic hyperlinking in a MID. Dynamic hyperlinking is the essence of advanced IETM capability. The element types are:

- **get** - a link to a location element found in a <locContainer>, <poolContainer> or directly to an < **infoContainer**> in the MID document instance. ***Note: In the current version of the MID,* locContainer *and* poolContainer *functions have been incorporated in the pool element.***
- #PCDATA - directly displayable content.
- **script** - A MID script that returns a node of information based on the evaluation of conditions defined within the script

Element types that have this content model can be linked to external databases through location elements or to internal containers. When the "get" link points to a location model in a HyTime location address, this method of indirect linking is referred to as a "location ladder".

When information from an external source is copied into a MID document instance, this is called a "hard MID". A MID that accesses an external document database through the use of HyTime queries, called HyQs or other HyTime location element types is referred to as a "soft" MID. The following are some reasons for creating "soft" and "hard" MIDs:

- A MID document instance is delivered without the supporting database. This is a "hard" MID in which all of the location elements are resolved and the content that they locate is copied directly into the MID document instance. This could be done because the target Presentation System cannot interpret and execute queries.
- A MID document is created without a supporting database. In this "hard" MID, there is no requirement for the external database; so, the author creates information directly in the MID form. Such information includes general maintenance procedures that are reused across systems without change.
- A MID document instance is delivered with the supporting database. This "soft" MID might be preferable because the database contains information not duplicated or referenced by the IETM that must be preserved without alteration.
- A "soft" MID document instance is maintained as the source database is undergoing changes that must be reflected in the contents of the MID. This would be typical of production environments that create IETMs for delivery, e.g., integrated logistics support groups. For example, the MIL-D-87269 database contains technical information about systems with long life cycles. The frequency for updating information in the database is rapid in early stages of system development. Soft MIDs used for the early stages of development allow an author to use the MID Presentation System as an integrating tool which is slowly hardened as design information is configured.

Note that the degree of hardening can vary among and within MID document instances. Any MID can contain a mixture of "soft" and "hard" addresses. How this mix is defined depends on the mission of the MID within the contractual deliverables of a weapons system.

C.3.1.6 Using Location Models

Below is an example of "soft" linking. In the example, a **HyQ** is shown along with a "hard" link for comparison.

```
<nameloc id="i24761144.name">
   <HyQ qdomain="i24761144">Proploc( DOMROOT ATTVAL[name] EMPTY)
 <nameloc id="i24761144">
   <nmlist docorsub="house">i24761144
 <treeloc id="i236438177.fc" locsrc="i236438177"> 1 1
  <nmlist docorsub="house">i236438177
```

Three types of "soft" links are illustrated:

a. Named Location with Query - upon access, the **HyQ** query is evaluated and the string inside the attribute value "name" is returned. The query domain (qdomain) value (i24761144) is used to determine the target of the query operation (i.e., the element or entity where the value is located).

b. Named Location with Name List - upon access, this opens the document or subdocument pointed to in the location of a document referenced by the "house" entity. The ID of the element to be located in the "house" document is the content of the element, i24761144.

c. Tree Location with Name List - upon access, this also opens the "house" document and locates the element with the ID value "i236438177". Upon locating the element, the markers (1 1) are used to count elements by their

position in the element hierarchy until the desired element is located. This method is used to locate information in a node not identified by an ID or other named value.

Other types of HyTime location models are also supported by the MID.

Within the Master Script or inside an < **infoContainer**>, an element like the following is used to access a location element:

```
<infoContainer id=welcome>
  <title><get target=i24761144.name></title>
```

When encountered in the < **infoContainer**>, the <**get**> accesses the <**nameloc**> whose ID matches the value in the <**get**> target attribute, (target=124761144.name). This is the first **nameloc** in the example above (a.). It contains a **HyQ** query to open the MIL-D-87269 document instance whose location is encoded in an entity declaration "house". That entity is accessed through the next **nameloc** whose ID is "i24761144". This **nameloc** contains a name list <**nmlist**> which has a "document or subdocument" attribute whose value, "house" is the actual name of the entity that locates the "house" document which contains an element whose ID is also "i24761144".

When the value in the "house" document is returned, it replaces the < **get**> link in the MID as the content of the < **title**> element. At that point, if simply displayed, the link remains soft; however, if the value is actually copied into the location of the <**get**>, the link becomes "hard". That is, the content of the < **title**> element is no longer a link; it is text (#PCDATA) displayed as the title.

The technique of using queries is referred to as "late binding". It enables an author to postpone the actual insertion of content into an IETM until some event in the project schedule indicates that the content is ready. By using a script and query mechanisms to evaluate conditions of other documents, dynamic linking and late binding concepts can be used in powerful and flexible ways.

While this location ladder is complex at first glance, it illustrates these concepts:

- All external locations are identified by entity declarations
- The name space of each SGML document instance is encapsulated in scope. Therefore, having a < **nameloc**> in one instance with an ID will not conflict with an element in another instance with the same value for its ID.
- Location ladders and indirect linking enable powerful reuse of links.

A benefit of using the indirect location element types is that once the location of an object is encoded with a HyTime location address, other elements can use the same location element to locate the same data from anywhere in a MID document instance. A non-redundant location scheme reduces the maintenance of a MID and enables the reuse of linking information.

C.3.2 Using Non-MIL-D-87269 or MIL-M-87268 Specifications for MID Designs

The MID DTD enables MIDs to be defined and delivered for sources of data other than the MIL-D-87269 class of document type definitions and non-SGML data types. Because the source document types are decoupled from the MID components that provide the description of the Presentation System, and these can be decoupled from the custom libraries (e.g., **HyQ** functions or other access methods) that access the source document, it is possible with careful design to provide highly reusable components.

The MID presentation components apply to source documents that use a traditional paper-based approach. If the source document type is changed (e.g., World Wide Web Hypertext Markup Language (HTML), the Computer Graphics Metafile (CGM), semi-conductor manufacturing embedded training), only the SGML elements containing the information directly related to the document type is modified, e.g., the queries. If a different database type (e.g., relational) is required, notation locations or external processes, (e.q., SQL), for queries in the new database type are used.